ソフトウェア・シンポジウム 2016 in 米子

論文集

ソフトウェア技術者協会

ソフトウェア・シンポジウムは，ソフトウェア技術に関わるさまざまな人びと，技術者，研究者，教育者，学生などが一堂に集い，発表や議論を通じて互いの経験や成果を共有することを目的に，毎年全国各地で開催しています．

第 36 回目を迎える 2016 年のソフトウェア・シンポジウムでは，SS2015 から始めたフォーラムセッションと Future Presentation（※） という発表形式をさらに発展させたものにしたいと考えています．このほか，SS2015 に引き続き，論文発表や事例報告と，ワーキンググループで議論を行います．また，ワーキンググループと並行して，チュートリアルセッションも 2 つ用意しました．基本から学び，実務や研究に役立てたいという方はぜひご利用ください．

今回は，“魅力的な「ソフトウェア開発」を目指して”をテーマに，さまざまな観点からの発表や講演を行います．ソフトウェア開発について，これまでの延長ではない捉え方をしていただく機会として，是非会場に足を運んでいただき，情報交換と深い議論をする場として活用してください！ソフトウェア・シンポジウムは，ソフトウェア技術に関わるさまざまな人びと，技術者，研究者，教育者，学生などが一堂に集い，発表や議論を通じて互いの経験や成果を共有することを目的に，毎年全国各地で開催しています．

※ ソフトウェア開発技術の進化に直接的，間接的に役立つと思われる，「先進的なアイデア」や「さまざまな技術や経験を融合した提案」などを議論するセッション．

開催概要

■ 日程 : 2016 年 6 月 5 日（日曜日）～ 8 日（水曜日）
6 月 5 日（日曜日）: 併設イベント
6 月 6 日（月曜日）～ 8 日（水曜日）: 本会議

■ 場所（併設イベント）: 米子コンベンションセンター BIG SHIP
情報プラザ（エントランスホール・1 階）

■ 場所（本会議）: 米子コンベンションセンター BIG SHIP

■ 主催 : ソフトウェア技術者協会

■ 後援 : 情報処理推進機構

■ 協賛 : 鳥取県情報産業協会, 米子工業高等専門学校, IT 記者会,
アジャイルプロセス協議会, オープンソースソフトウェア協会,
情報サービス産業協会, 情報処理学会, ソフトウェアテスト技術振興協会,
ソフトウェア・メインテナンス研究会, 電子情報通信学会,
日本情報システム・ユーザー協会, 日本ソフトウェア科学会,
組込みシステム技術協会, 日本 SPI コンソーシアム,
日本ファンクションポイントユーザ会, 派生開発推進協議会,
日本科学技術連盟, 組込みソフトウェア管理者・技術者育成研究会,
TOPPERS プロジェクト, PMI 日本支部, アドバンスト・ビジネス創造協会

スタッフ一覧

実行委員会

実行委員長

宮田 一平（アイエックス・ナレッジ）

松本 健一（奈良先端科学技術大学院大学）

実行委員

鯵坂 恒夫（和歌山大学）

伊藤 昌夫（ニルソフトウェア）

小笠原 秀人（東芝）

岸田 孝一（SRA）

栗田 太郎（ソニー）

小松 久美子（帝塚山学院大学）

杉田 義明（福善上海）

鈴木 裕信（鈴木裕信事務所）

田中 美穂（アートシステム）

中野 秀男（帝塚山学院大学）

奈良 隆正（NARA コンサルティング）

野村 行憲（アイシーエス）

プログラム委員会

プログラム委員長

小笠原 秀人（東芝）

大平 雅雄（和歌山大学）

プログラム委員

秋山 浩一（富士ゼロックス）

鯵坂 恒夫（和歌山大学）

安達 賢二（HBA）

天嵜 聡介（岡山県立大学）

荒木 啓二郎（九州大学）

伊藤 昌夫（ニルソフトウェア）

岩崎 孝司（富士通九州ネットワークテクノロジーズ）

臼杵 誠（富士通）

落水 浩一郎（金沢工業大学）

小田 朋宏（SRA）

片山 徹郎（宮崎大学）

北須賀 輝明（熊本大学）

日下部 茂（九州大学）

楠本 真二（大阪大学）

栗田 太郎（ソニー，フェリカネットワークス）

小嶋 勉（DNV ビジネス・アシュアランス・ジャパン）

後藤 徳彦（NEC ソリューションイノベータ）

古畑 慶次（デンソー技研センター）

阪井 誠（SRA）

酒匂 寛（デザイナーズデン）

佐原 伸（法政大学）

小林 修（SRA）

鈴木 裕信（鈴木裕信事務所）

鈴木 正人（北陸先端科学技術大学院大学）

高木 智彦（香川大学）

高橋 芳広（トリプル・アイ企画）
田中 康（東京工業大学）
張 漢明（南山大学）
土肥 正（広島大学）
富松 篤典（電盛社）
中谷 多哉子（放送大学）
中森 博晃（パナソニック ファクトリーソリューションズ）
西 康晴（電気通信大学）
野村 行憲（アイシーエス）
野呂 昌満（南山大学）
端山 毅（エヌ・ティ・ティ・データ）
本多 慶匡（東京エレクトロン）
松尾谷 徹（デバッグ工学研究所）
松本 健一（奈良先端科学技術大学院大学）
水野 修（京都工芸繊維大学）
宮本 祐子（宇宙航空研究開発機構）
宗平 順己（ロックオン）
森崎 修司（名古屋大学）
諸岡 隆司（中電シーティーアイ）
八木 将計（日立製作所）
山本 修一郎（名古屋大学）
米島 博司（パフォーマンス・インプルーブメント・アソシエイツ）
劉 少英（法政大学）

事務局

栗田 太郎（ソニー，フェリカネットワークス）

ソフトウェア・シンポジウム 2016 in 米子 目次

VDM-SL 仕様からの Smalltalk プログラムの自動生成

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

荒木 啓二郎
九州大学
araki@csce.kyushu-u.ac.jp

要旨

VDM-SL は実行可能なサブセットを持ち，複数のインタプリタ実装が提供されている形式仕様記述言語である．本稿は，*Smalltalk* 環境上に構築された *ViennaTalk* ライブラリに実装された，*VDM-SL* による実行可能仕様からの *Smalltalk* プログラムの自動生成器の実装を説明し，生成された *Smalltalk* ソースコードの性能評価を示す．

1. はじめに

ソフトウェアシステムの仕様を形式的に記述することで 仕様記述の問題点をより早期に発見し修正する開発手法として, 形式手法が注目されている [1, 2]．形式仕様記述言語 VDM-SL [3] は実行可能なサブセットを持ち，複数のインタプリタ実装が提供されている [4]．実行可能な VDM 仕様は vdmUnit 等のテストフレームワークを利用して単体テストを継続的に行うことで，高い品質の仕様記述が可能であることが事例から知られている [5].

筆者らはこれまで VDM-SL インタプリタを Smalltalk 環境から利用するためのクラスライブラリ ViennaTalk を開発し，仕様の妥当性を確認するための UI プロトタイピング環境 Lively Walk-Through [6], や Web API プロトタイプ環境 Webly Walk-Through[6, 7] および，WebIDE サーバ VDMPad [8, 9] を構築してきた．ViennaTalk は Smalltalk 環境の柔軟性を活かして，VDM-SL で記述された実行可能仕様を Smalltalk 環境の一部として利用するための Smalltalk ライブラリである．ViennaTalk は 既存の VDM-SL インタプリタを外部インタプリタとして利用し，Smalltalk プログラムの動作の一部として VDM-SL 仕様を実行する仕組みを提供することで，VDM-SL のための開発ツールを開発することを容易にする．

本稿では、ViennaTalk に実装した，VDM-SL 仕様から Smalltalk プログラムを自動生成する機能について述べる．VDM-SL 仕様から Smalltalk プログラムを自動生成し，他の Smalltalk プログラムと連携して動作させることで，インタプリタとの連携に起因するオーバーヘッドを減らし，インタプリタ実行よりも実行効率のよい連携を可能にした．

本稿では，VDM-SL の概要を紹介し，自動生成の概要，特に VDM の特徴でもある強力な集合演算，写像，およびパターンマッチの実現について説明する．生成されたプログラムの性能評価として，エラトステネスの篩のアルゴリズムを列を多用した記述，集合を多用した記述，および写像を多用した記述を作成し，それぞれについて処理時間を VDMTools [10] および VDMJ [11] のインタプリタ実装と比較した．VDMTools および VDMJ はともに実際のソフトウェア開発で利用された実績のあるインタプリタ実装である．

2. VDM-SL の概要

VDM-SL はモデル規範型の形式仕様記述言語で，記述対象を抽象するための型，定数，関数を定義し，状態空間の定義と状態を変更する操作を定義することで，システムの入出力や状態変化を記述する．インタプリタによって実行可能なサブセットを持つが，プログラミング言語とは異なり，仕様記述は動作させることが目的ではなく，システムの動作における入出力や状態変化を規定することを目的として記述する．また，インタプリタは実行すること自体が目的ではなく，記述されたシステムの動作をシミュレートすることでその振る舞いを理解し確認することが目的である．

VDM-SL にはモジュール機能があるが，本稿では簡単のためモジュールを持たない記述について説明す

る．VDM-SL による仕様記述は types で始まる型定義部，values で始まる定数定義部，functions で始まる関数定義部，state で始まる状態定義部，および operations で始まる操作定義部からなる．各定義部は同種の定義部が複数存在しても良く，また，必要がなければ省略しても良い．

VDM-SL で提供されている型には，自然数型，非 0 自然数型，整数型，実数型，文字型，ブール型，引用型，トークン型などの基本型と，複合型，列型，集合型，直積型，合併型，オプション型，写像型などの合成型がある．文字列のための型はなく，文字列は文字型の列 seq of char として扱う．

2.1. エラトステネスの篩と双子素数

エラトステネスの篩は素数列を求める古典的なアルゴリズムである．2 から N までの間にある素数を求める場合，まずは素数候補となる 2 から N までの自然数の昇順の列を定義する．そして，候補列から先頭要素を取り出した数を素数として素数の列に加え，その全ての倍数を候補列から削除する操作を，候補列が空になるまで続けることで，素数列を得ることができる．双子素数とは，隣り合う 2 つの奇数がともに素数となっているペアである．例えば，3 と 5，5 と 7，11 と 13 は双子素数である．

図 1 に，素数を求めるエラトステネスの篩と，双子素数を求めるアルゴリズムを VDM-SL で記述した実行可能な仕様を示す．この仕様には型定義部および定数定義部はなく，状態定義部，関数定義部，および 2 つの操作定義部から成っている．

図 1 の状態定義部では，候補列 space および 素数列 primes の 2 つの変数が非 0 自然数の列として定義されている．init 句では，初期値として共に空列を定義している．

続いて operations で始まる操作定義部では，setup, next, sieve の 3 つの操作が定義されている．setup 操作は非 0 自然数を引数として取り，候補列 space に 2 から引数の値までの自然数の昇順の列，素数列 primes に空列 [] をそれぞれ代入する．next 操作は候補列 space が空かどうかを確認し，空の場合には 終了をあらわすために nil を返し，空でない場合には先頭要素 hd space を素数列 primes の最後尾に連結し，篩にかける．篩をかける sieve 操作は，列の内包表記を使って候補列 space から引数で与えられた数

```
state Eratosthenes of
    space : seq of nat1
    primes : seq of nat1
init s == s = mk_Eratosthenes([], [])
end
operations
    setup : nat1 ==> ()
    setup(x) ==
        (space := [i | i in set {2, ..., x}];
        primes := []);

    next : () ==> [nat1]
    next() ==
        if
            space = []
        then
            return nil
        else
            (dcl x:nat1 := hd space;
            space := tl space;
            primes := primes ^ [x];
            sieve(x);
            return x);

    sieve : nat1 ==> ()
    sieve(x) ==
        space :=
            [space(i) |
            i in set inds space &
            space(i) mod x <> 0];
functions
    twins : seq of nat1 -> seq of (nat1* nat1)
    twins([x1, x2] ^ rest) ==
        let ts =
            if rest = []
            then []
            else twins([x2] ^ rest)
        in
            if x2 - x1 = 2
            then [mk_(x1, x2)] ^ ts
            else ts;
operations
    prime10000 : () ==> seq of nat1
    prime10000() ==
        (setup(10000);
        while next() <> nil do skip;
        return primes);

    twinprime10000 : () ==> seq of (nat1* nat1)
    twinprime10000() ==
        (setup(10000);
        while next() <> nil do skip;
        return twins(primes));
```

図 1. エラトステネスの篩の VDM-SL 仕様

で割った剰余が 0 でない，すなわち引数の倍数でない要素のみからなる列を作成し，それを新たな候補列 space としている．

functions で始まる関数定義部では，非 0 自然数列から隣り合う要素同士の差が 2 となるペアを列にして取り出す関数 twins がパターンマッチを使って定義されている．パターンマッチは関数 twins の仮引数 [x1, x2] ^ rest で使われている．これは，2 要素の列にマッチし第 1 要素を x，第 2 要素を y とする列の列挙パターンと，それと任意の列にマッチし rest とするパターンを連結した列連結パターンであり，rest には列の列挙パターンの残り部分である第 3 要素以降の部分列が束縛される．

最後に 2 番目の操作定義部では，それぞれ 10000 以下の素数列と双子素数列を求める操作 prime10000 と twinprime10000 が定義されている．

3. 自動生成

筆者らは Smalltalk プログラムと VDM-SL 仕様の連携を重視して ViennaTalk を開発してきた．コード生成においても，Smalltalk らしい表現をしたプログラムを生成すること，生成元の VDM-SL 仕様と文面上の対応関係がわかりやすいことを目標として機能設計し実装した．そのために既存のクラスやメソッドに適合するように，また，生成された Smalltalk プログラムの中で使われているメッセージ名から元の VDM-SL 仕様での記述が類推しやすいように，Smalltalk 基本ライブラリを拡張した．

本節では，VDM-SL と Smalltalk の間の型システムの違いを埋めるための対応関係，VDM-SL の組み込み関数のための Smalltalk ライブラリへの拡張，パターンマッチングや不変条件のような Smalltalk にはない言語機能の実現について説明する．

3.1. VDM-SL の型および値と Smalltalk オブジェクトの対応関係

VDM-SL と Smalltalk では型システムが大きく異なる．Smalltalk はクラスベースの動的型付き言語であり，VDM-SL を含む静的型付き言語での型に相当する言語要素としてクラスが挙げられる．VDM-SL では 1 つの値が同時に複数の型の値であり得るが，Smalltalk では 1 つのオブジェクトは 1 つのクラスに属するため，VDM-SL の型を Smalltalk のクラスに対応付けることは適切ではないと考えた．ViennaTalk でのコード生成では VDM-SL の型をクラスではなく，型を表現するオブジェクトに対応付けることとし，型を表現するためのクラスライブラリを実装した．ViennaTalk では VDM の型もオブジェクトであり，型オブジェクトにメッセージを送ることで新たな型オブジェクトを合成したり，また，型の間の関係や型と値の関係についての問い合わせを記述できる．

VDM-SL の値を Smalltalk で実現するにあたって，ViennaTalk では可能なかぎり Smalltalk の既存のクラスを利用した．VDM-SL の型システムでは，整数型の値は全て実数型の値でもある点が Smalltalk を含む多くのプログラミング言語と異なる．すなわち，値 1 は Smaltalk では SmallInteger クラスのインスタンスだが，VDM-SL では 1 は nat, nat1, int, real の 4 つの型全ての値である．ViennaTalk では SmallInteger クラスと VDM-SL の nat 型を直接対応付けるのではなく，1 というインスタンスを nat 型の Smalltalk 側の実装である ViennaNatType クラスのインスタンスに対応付ける．ViennaNatType クラスについては次節で説明する．

VDM-SL の型システムの特徴としては，型にはその値が満たすべき不変条件を付与することができる点が挙げられる．例えば，types nonzero = int inv x == x <> 0 と定義することで，非 0 整数の型 nonzero を定義することができる．

VDMTools での VDM-SL からの C++コード生成では，VDM-SL の型に対応するクラスを定義している．ViennaTalk では Smalltalk の既存のクラスを利用することで，Smalltalk での整数オブジェクトはそのまま VDM-SL での整数値として使え，また，VDM-SL での集合はそのまま Smalltalk でも Smalltalk の集合オブジェクトとして使うことができる．

VDM-SL の型と，その型を表現する Smalltalk 表現式およびそのクラス，その型の値のクラスの対応関係を表 1 に示す．

3.2. Smalltalk ライブラリの拡張

VDM-SL の特徴としては，集合や写像など抽象度の高い型と強力な組み込み機能が挙げられる．Smalltalk は豊かなクラスライブラリを持っており，集合や写像を扱う抽象度も VDM-SL と大きな違いはないが，いくつかの機能を既存のクラスに追加する必要があった．コード

表 1. VDM-SL の型とその値に対応する Smalltalk での表現式とクラス

VDM の型	VDM 表現	Smalltalk 表現	型のクラス	値のクラス
自然数	nat	ViennaType nat	ViennaNatType	Integer
非 0 自然数	nat1	ViennaType nat1	ViennaNat1Type	Integer
整数	int	ViennaType int	ViennaIntType	Integer
実数	real	ViennaType real	ViennaRealType	Float
ブール型	bool	ViennaType bool	ViennaBoolType	Boolean
引用型	<quote>	ViennaType quote: *#quote*	ViennaQuoteType	Symbol
オプション型	[*t*]	*t* optional	ViennaOptionType	*t* のクラスまたは UndefinedObject
直積型	*t1* * *t2*	*t1* * *t2*	ViennaProductType	Array
合併型	*t1* | *t2*	*t1* | *t2*	ViennaUnionType	*t1* または *t2* のクラス
集合型	set of *t*	*t* set	ViennaSetType	Set
列型	seq of *t*	*t* seq	ViennaSeqType	OrderedCollection
非空列型	seq1 of *t*	*t* seq1	ViennaSeq1Type	OrderedCollection
写像型	map *t1* to *t2*	*t1* mapTo: *t2*	ViennaMapType	Dictionary
単射型	inmap *t1* to *t2*	*t1* inmapTo: *t2*	ViennaInmapType	Dictionary
部分関数型	*t1* -> *t2*	*t1* -> *t2*	ViennaPartialFunctionType	BlockClosure
全関数型	*t1* +> *t2*	*t1* +> *t2*	ViennaTotalFunctionType	BlockClosure
複合型	compose *t* of *f1* : *t1* *f2* :- *t2* *t3* end	ViennaType compose: '*t*' of: {*f1* . false . *t1*. {*f2* . true . *t2*}. {nil . false . *t3*}}	ViennaCompositeType	ViennaComposite
不変条件	*t* inv *pattern* ==*expr*	*t* inv: [:*v* | *expr*]	ViennaConstrainedType	*t* のクラス

表 2. 既存クラスに対するコード生成のための主な拡張

既存クラス名	メソッド名	機能
Object	viennaString	対応する VDM-SL 表現式を得る
Context	viennaReturn: *value*	関数や操作のコンテキストから return する
Collection	onlyOneSatisfy: *block* power powerDo: *block*	*block* の評価結果が真になる要素を 1 つだけ持つかを判定する 冪集合を得る 全ての部分集合を列挙してそれぞれを引数として *block* を評価する
SequenceableCollection, Dictionary, BlockClosure	applyTo: *value*	*value* 番目の要素を得る
Dictionary, BlockClosure	comp: *map-or-func* ** *value*	関数合成を得る *value* 回繰り返し関数合成を行った結果を得る

表 3. コード生成のための新規クラス

追加クラス	機能
ViennaType 及び サブクラス	VDM-SL の型
ViennaComposite	複合型の値
ViennaToken	トークン型の値
ViennaComposition	関数合成の結果
ViennaIteration	繰り返し関数合成の結果

生成のための既存クラスへの拡張を図 2 に示す．この拡張により，VDM-SL が定義する全ての組み込み関数について，Smalltalk オブジェクトが提供する機能として実現された．

第 3.1 節で説明した通り，VDM-SL の型を表すオブジェクトを新規にクラスとして定義した．また，複合型およびトークン型は Smalltalk の標準クラスライブラリに直接対応するクラスがなかったために，それぞれの型の値を表現するためのクラスを定義した．関数合成や繰り返し関数合成については，既存のクラスの拡張として一般化した定義が困難であるため，合成結果を表現するクラスを定義した．表 3 にまとめて示す．

3.3. パターンマッチング

VDM-SL では強力なパターンマッチメカニズムが提供されている．パターンマッチは関数定義，λ式，集合内包表記等，仮パラメータを扱う多くの場面で利用することができる．例えば，同じ部分列が 2 度繰り返す列のみを引数として受け付け，その部分列を返す関数を定義する VDM-SL のλ式を以下に示す．

```
(lambda x^x:seq of char & x)
```

上記のλ式で，^は連結パターンと呼ばれ，マッチ対象となる列に対するあらゆる分割の中からそれぞれの部分列が 2 つのパターンにマッチする分割を得る．x ^ x は，引数として与えられた列に対して，2 つの部分列に分割してそれぞれを同一パターン x にマッチさせることで，同じ部分列の繰り返しに対してマッチする．上記のλ式に [1,2,3,1,2,3] を引数として関数適用すると [1,2,3] が得られる．また，[1,2,1,2] を引数として関数適用すると，[1,2] が得られる．同様に集合の分割に対してマッチさせる和集合パターンや，写像の分割に対してマッチさせる和写像パターンがある．

ViennaTalk では，自動生成された Smalltalk のランタイム支援を行う ViennaUtil クラスに各種パターンマッチをさせる機能を実装し，上記のλ式からは以下の Smalltalk プログラムが生成される．

```
[ :_lmd |
| x |
(((ViennaRuntimeUtil
  matchTuple:
    {(ViennaRuntimeUtil
      match: (ViennaRuntimeUtil matchIdentifier: 'x')
      conc: (ViennaRuntimeUtil matchIdentifier: 'x'))})
  value: {_lmd})
  ifEmpty: [ false ]
  ifNotEmpty: [ :binds |
    x ← binds first at: 'x'.
    true ])
  ifFalse: [ ViennaNoMatch signal ].
x ]
```

自動生成されたプログラムでは，パターンマッチの処理を行う部分が長くなり，見通しが悪くなっている．クロージャは VDM-SL のλ式には出ていない仮引数 _lmd を宣言している．そして渡されたオブジェクトに対して，ViennaRuntimeUtil を利用してネストしたパターンマッチ処理を行っている．マッチに成功した場合にはマッチした束縛環境 binds から x を取り出し，一時変数 x に代入しλ式の本体部分である x を評価する．マッチに失敗した場合には，ViennaNoMatch 例外を投げる．

上記の処理のために，パターンマッチを含む VDM-SL 仕様から生成されたプログラムは Smalltalk プログラムとしての可読性に問題がある．実際のコード生成では下記の VDM-SL のλ式での x のように識別子のみからなる単純なパターンについては，生成されたプログラムの性能や Smalltalk プログラムらしさの点から，パターンマッチを介さずにクロージャの引数として扱うなどをしている．

```
(lambda x:int & x*2)
```

生成された Smalltalk プログラムを以下に示す．

```
[ :x | x * 2 ]
```

マッチング処理を介さずに直接クロージャの引数として生成される．VDM-SL のλ式との対応関係を容易に把握することができる．

3.4. 不変条件，事前条件，事後条件，メジャー関数

VDM-SL では型および状態に対して不変条件，関数および操作に対して事前条件および事後条件，さらに再帰

関数に対して再帰の停止性の確認を支援するためのメジャー関数を定義することができる．型に対する不変条件は第 3.1 節で説明した通り，ViennaTalk では型オブジェクトに対して inv: *block* により定義することができる．

状態の不変条件についても，Smalltalk クラスを生成する場合には，生成されたプログラムにおいても不変条件が実行時検査される．状態の不変状態は，生成された Smalltalk クラスでは，inv メソッドとして定義される．Smalltalk の Slot と呼ばれる機能を利用して，VDM-SL の操作から生成された Smalltalk メソッドをコンパイルする際に，代入後に inv メソッドを実行する命令を Smalltalk メソッドのバイトコードに埋め込む．

関数および操作の事前条件および事後条件，および再帰関数のメジャー関数については，プログラムコードの生成と実行という目的においては必要ないと考え，実装していない．コード生成の目的はあくまでプログラムコードの実行であり，仕様そのものの健全性は生成されたコードの実行時の検査ではなく，仕様記述からの証明責務の生成およびインタプリタ実行によって確認されるべきである．

型の不変条件については，実行時の振る舞いに影響を与えるためにコード生成に実装した．具体的には，ある値がある型に属するかどうかを検査する VDM-SL の言語機能として is 式がある．この is 式の振る舞いを正しくコード生成に反映させるためには，型に付けられた不変条件が必要である．また，集合や列，写像の内包表現や forall 等の量化式，let-be 式や let-be 文などの型束縛による値の生成においても，仕様と一致した動作をするプログラムを生成するためには型につけられた不変条件が必要である．

4. 評価

ViennaTalk によって生成されたプログラムの性能評価として，第 2.1 節で示したエラトステネスの篩による双子素数を求める VDM-SL 仕様から生成されたプログラムの実行時間を計測した．比較対象としては，代表的なインタプリタ実装である VDMTools と VDMJ を選んだ．

また，図 1 に示した仕様記述では主に列型を使っているが，VDM-SL の仕様で列型と同様に頻繁に利用される集合型および写像型を使ったエラトステネスの篩の仕様を記述し，Smalltalk プログラムを生成して実行時間を計測した．VDM インタプリタおよび Smalltalk 環境の実行環境は以下の通りである．

```
CPU     Intel Core i5 2.5 GHz
主記憶  16GB 1600MHz DDR3
OS      Mac OS X 10.11.3 (El Capitan)
```

表 4. 列型を利用したエラトステネスの篩と双子素数の計算時間 (単位は ms)

	VT	VDMJ	VDMTools	ST
prime10000	683	4,143	24,569	40
twinprime10000	750	4,475	24,991	40

VT は ViennaTalk による生成
ST は人手で改善した Smalltalk プログラム

4.1. 列操作

図 1 の VDM-SL 仕様から生成された Smalltalk プログラムを図 2 に示す．パターンマッチ部分を除くと通常の Smalltalk プログラムとして理解可能なプログラムが生成されている．ただし，人が書いた Smalltalk プログラムと比べると，コレクションオブジェクトの不要なコピーなどがあり，人手による改善の余地が残っている．生成された Smalltalk プログラムを人手で改善したプログラムを図 3 に示す．

改善されたプログラムは，生成されたプログラムが利用しているクラスおよび制御構造に対して，より Smalltalk プログラムとして適切であると考えられるクラスおよび制御構造を使って実装された．例えば，生成されたプログラム（図 2）では space は OrderedCollection クラスのオブジェクトが使われているのに対して，人手で改良されたプログラム（図 3）では Array クラスのオブジェクトが使われており，より高速な処理となっている．

また，双子素数を発見する関数 twins については，パターンマッチではなく，要素を 1 つずつずらした列を同時に列挙してそれぞれの列からの値を処理することで，高速な処理を実現している．

ViennaTalk で生成された Smalltalk プログラム，VDMJ によるインタプリタ実行，VDMTools によるインタプリタ実行および人手で改善した Smalltalk プログ

```
| Eratosthenes next primes init_Eratosthenes
 space twins sieve setup twinprime10000
  prime10000 |
twins ← [ :_func |
| x1 x2 rest |
(((ViennaRuntimeUtil
  matchTuple:
    {(ViennaRuntimeUtil
      match:
        (ViennaRuntimeUtil
          matchSequenceEnumeration:
            {(ViennaRuntimeUtil matchIdentifier: 'x1').
            (ViennaRuntimeUtil matchIdentifier: 'x2')})
      conc: (ViennaRuntimeUtil matchIdentifier: 'rest')
      left: 2)}) value: {_func})
  ifEmpty: [ false ]
  ifNotEmpty: [ :binds |
    x1 ← binds first at: 'x1'.
    x2 ← binds first at: 'x2'.
    rest ← binds first at: 'rest'.
    true ])
  ifFalse: [ ViennaNoMatch signal ].
[ | ts |
ts ← rest = {} asOrderedCollection
  ifTrue: [ {} asOrderedCollection ]
  ifFalse: [ twins
    applyTo: {({x2} asOrderedCollection , rest)} ].
x2 - x1 = 2
  ifTrue: [ {{x1. x2}} asOrderedCollection , ts ]
  ifFalse: [ ts ] ] value ].
setup ← [ :x |
[ space ← ((2 to: x) collect: [ :i | i ])
    asOrderedCollection.
primes ← {} asOrderedCollection ] value ].
next ← [
space = {} asOrderedCollection
  ifTrue: [ thisContext viennaReturn: nil ]
  ifFalse: [
    [ | x |
    x ← space first.
    space ← space tail.
    primes ← primes , {x} asOrderedCollection.
    sieve valueWithArguments: {x}.
    thisContext viennaReturn: x ] value ] ].
sieve ← [ :x |
space ← ((1 to: space size)
  select: [ :i | (space applyTo: {i}) \\ x ~= 0 ]
  thenCollect: [ :i | space applyTo: {i} ])
  asOrderedCollection ].
prime10000 ← [
[ setup valueWithArguments: {10000}.
[ (next applyTo: {}) ~= nil ] whileTrue: [  ].
thisContext
  viennaReturn: primes) ] value ].
twinprime10000 ← [
[ setup valueWithArguments: {10000}.
[ (next applyTo: {}) ~= nil ] whileTrue: [  ].
thisContext
  viennaReturn: (twins applyTo: {primes}) ] value ].
space ← {} asOrderedCollection.
primes ← {} asOrderedCollection.
Eratosthenes ← ViennaCompositeType
  constructorName: 'Eratosthenes'
  withAll:
    {{'space'. false. (ViennaType nat1 seq)}.
    {'primes'. false. (ViennaType nat1 seq)}}.
init_Eratosthenes ← [ :s |
s  = (Eratosthenes
  applyTo:
    {({} asOrderedCollection).
     ({} asOrderedCollection)}) ].
```

図 2. 列型を使ったエラトステネスの篩の仕様から生成された Smalltalk プログラム

```
|  next primes  space twins sieve
setup twinprime10000 prime10000 |
twins ← [ :xs |
  Array new: 1024 streamContents: [:stream |
    (xs copyFrom: 1 to: xs size - 1)
      with: xs copyWithoutFirst
      do: [ :x :y |
        y - x = 2
          ifTrue: [stream nextPut: {x. y}]]]].
setup ← [ :x |
space ← (2 to: x) asArray.
primes ←OrderedCollection new].
next ← [ space
  ifEmpty: [ nil ]
  ifNotEmpty: [ | x |
    x ← space first.
    sieve value: x.
    primes add: x]].
sieve ← [ :x |
space ← space select: [ :y| y \\ x > 0]].

prime10000 ← [
setup value: 10000.
[next value notNil ] whileTrue.
primes ].
twinprime10000 ← [
setup value: 10000.
[ next value notNil ] whileTrue: [  ].
twins value: primes ].
space ← {}.
primes ← OrderedCollection new.
```

図 3. 生成されたプログラム（図 2）を人手で改善した Smalltalk プログラム

ラムによる prime1000 および twinprime1000 の実行時間を表 4 に示す．VDMJ および VDMTools によるインタプリタ実行に対しては優位な処理速度が実現されている．人手による Smalltalk プログラムと比較すると prime10000 で 17 倍以上，twinprime10000 で 18 倍以上の実行時間となっており，生成について改善の余地が残されている．

4.2. 集合操作

space および primes を列型ではなく集合型を使って，同様の仕様を記述した．VDM-SL 仕様のほとんどは列型の機能を対応する集合型の機能に変更することで記述することができたが，大きな変更をおこなった部分を図 4 に示す．twin 関数については，集合の内包表記を使ってより簡潔な記述になった．列を使った記述では自然数が昇順で並べられていたために候補列の先頭要素を取り出していたが，集合の場合には候補集合から最小値を求める必要があるため，min 関数を新たに定義した．min 関数は，引数のあらゆる値について x <= y となるような引数中の唯一の要素を返す．

ViennaTalk で自動生成した Smalltalk プログラ

```
functions
    twins : set of nat1 -> set of (nat1 * nat1)
    twins(xs) ==
        {mk_(x, y) |
         x, y in set xs &
         y - x = 2};
    min : set of nat1 -> nat1
    min(xs) ==
        iota x in set xs &
            (forall y in set xs & x <= y);
```

図 4. 集合型を使ったエラトステネスの篩と双子素数を求める VDM-SL 仕様

```
twins ← [ :xs |
[ :__set |
| _set |
_set ← Set new.
__set do: [ :x |
  __set do: [ :y |
    y - x = 2
      ifTrue: [ _set add: {x. y} ] ] ].
_set ] value: xs ].
min ← [ :xs |
xs
  detect: [ :x |
    [ :_forall |
     _forall allSatisfy: [ :y | x <= y ] ]
     value: xs ]
  ifNone: [ ViennaNoMatch signal ] ].
```

図 5. 集合型を使った twins 関数および min 関数の仕様から生成された Smalltalk プログラム

表 5. 集合型を利用したエラトステネスの篩と双子素数の計算時間 (単位は ms)

	VT	VDMJ	VDMTools
prime10000	788	109,029	N/A
twinprime10000	833	112,332	N/A

VT は ViennaTalk による生成
N/A は計算が 1 時間以内に終了しなかったことを示す

ムを図 5 に示す．図 4 の VDM-SL 仕様での twins 関数の定義では集合内包表記で 2 つの集合束縛された変数を使っているが，生成された Smalltalk プログラムは標準クラスライブラリでの通常のプログラミングで利用される機能を使って 2 重ループとして生成している．min 関数についても，detect:ifNone:や allSatisfy:といった Smalltalk プログラムで頻出する機能が使われている．自動生成された変数名は可読性に難があるが，プログラムの構成自体は人手による自然な Smalltalk プログラムに近いものが生成されている．

集合型で記述された prime10000 および twinprime10000 による性能評価を図 6 に示す．自動生成された Smalltalk プログラムは VDMJ インタプリタに対して大きく上回る処理速度となっている．VDMTools インタプリタでは 1 時間経過しても処理が終了しなかったため N/A とした．人手による改善では，使用クラスや制御構造を変更した結果，列型からの改善と同様のプログラムになる．人手によって改善されたプログラムの処理時間は表 4 を参照のこと．

4.3. 写像操作

space および primes を写像型を使って，同様の仕様を記述した．大きな変更をおこなった部分を図 6 に示す．写像を使った仕様では，space を素数候補となる非 0 自然数から候補として残っているかどうかへの写像として表現した．例えば，{2 |-> false, 3 |-> true, 4 |-> false} の場合，元の候補であった {2, 3, 4} のうち 3 が候補として残っていることを表す．primes は自然数 n から n 番目の素数 primes(n) への写像である．

ViennaTalk が生成した Smalltalk プログラムを図 7 に示す．実行時間を計測したが，2 から 10000 までの間にある素数を求まったのは 生成された Smalltalk プログラムのみであった．2 から 1000 までの間にある素数の計算に要した時間を計測した．

5. 議論

Smalltalk 上の VDM-SL 環境 ViennaTalk に実装された VDM-SL 仕様から Smalltalk プログラムの自動生成器について，双子素数を求めるアルゴリズム仕様によって既存のインタプリタ実装との性能を比較した．

```
functions
  twins : map nat1 to nat1
              -> set of (nat1 * nat1)
  twins(xs) ==
    {mk_(x, y) |
     x, y in set rng xs &
     y - x = 2};
  min : map nat1 to bool -> nat1
  min(xs) ==
    iota x in set dom xs &
      (xs(x) and forall y in set dom xs &
         xs(y) => x <= y);
```

図 6. 写像型を使ったエラトステネスの篩と双子素数を求める VDM-SL 仕様

```
twins ← [ :xs |
([ :__set |
 | _set |
_set ←Set new.
__set do: [ :x |
  __set do: [ :y |
    ((y - x) = 2)
      ifTrue: [_set add: {x . y} ]]].
_set] value: xs values asSet)].
min ← [ :xs |
(xs keys asSet
  detect: [ :x |
    ((xs applyTo: {x}) and:
      [([ :_forall |
          _forall allSatisfy: [ :y |
            ((xs applyTo: {y}) not or: [(x <= y)])]]
       value: xs keys asSet)])]
  ifNone: [ViennaNoMatch signal])].
```

図 7. 写像型を使った twins 関数および min 関数の仕様から生成された Smalltalk プログラム

表 6. 写像型を利用したエラトステネスの篩と双子素数の計算時間 (単位は ms)

	VT	VDMJ	VDMTools
prime10000	19,609	N/A	N/A
twinprime10000	19,903	N/A	N/A
prime1000	287	34,766	N/A
twinprime1000	312	35,043	N/A

VT は ViennaTalk による生成

N/A は計算が 1 時間以内に終了しなかったことを示す

列型を用いた仕様，集合型を用いた仕様，写像型を用いた仕様のいずれでも ViennaTalk は 既存のインタプリタ実装よりも実行速度の面で優位であった．

VDM-SL による実行可能仕様は実行すること自体が目的ではない．インタプリタは仕様記述の妥当性の確認および動作への理解を深めるためのツールであり，プログラムの自動生成は十分に妥当性および正当性が検査された仕様記述から実装を得るためのツールである．自動生成されたプログラムがインタプリタ実行よりも処理速度が優れていることは，必ずしもインタプリタよりも優れた処理系であるということではない．本稿での性能評価では，自動生成器が生成したプログラムを最終的なプログラム実装を得るための 1 つのステップとして位置付け，処理速度面においてインタプリタ実行よりも優れた性能が得られていることを示した．

表 6 に示した写像型を利用したエラトステネスの篩と双子素数では，prime10000 および twinprime10000 は自動生成器によるプログラムのみが計算を終了した．ViennaTalk の Smalltalk プログラム自動生成器は，VDM-SL 仕様のうち実用的に実行可能な領域を広げることで，ソフトウェア開発における VDM の有用性を高める一助となりうると考えられる．

自動生成されたプログラムの表現については，少なくとも Smalltalk プログラムとして読むに耐えるソースコードが得られた．しかし人が記述したプログラムとはまだ明確な差があり，改善の余地があると考えられる．操作および関数の事前条件事後条件，および，再帰関数のメジャー関数など，現在の実装では生成されたプログラムに反映されない部分があり，自動生成器の適用領域を広げるために，これらをプログラム自動生成に反映させることも今後の課題と考えられる．

6. まとめ

ViennaTalk が基礎としている Smalltalk 環境は柔軟性が高く，メタ記述能力の高い開発環境であり，ViennaTalk はその柔軟性とメタ記述能力を使って，VDM-SL のツールをより効率的に開発するためのメタ開発環境である．本稿で示した Smalltalk プログラム自動生成は，ViennaTalk 環境の中で，VDM-SL 仕様と Smalltalk 環境を繋ぐ橋渡しとしての役割を担っている．VDM-SL インタプリタとの連携に加えて コード生成が可能になることでより自由度の高いツール開発が

可能になることを期待し，今後も ViennaTalk の開発を継続する．

7. 謝辞

本研究の一部は JSPS 科研費 (24220001) および JSPS 科研費 (26330099) の助成を受けた．

参考文献

[1] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, ``Formal methods: Practice and experience,'' *ACM Computing Surveys*, vol. 41, no. 4, pp. 1--36, October 2009.

[2] N. Ubayashi, S. Nakajima, and M. Hirayama, ``Context-dependent product line engineering with lightweight formal approaches,'' *Sci. Comput. Program.*, vol. 78, no. 12, pp. 2331--2346, Dec. 2013.

[3] J. Fitzgerald and P. G. Larsen, *Modelling Systems -- Practical Tools and Techniques in Software Development*. Cambridge University Press, 1998, iSBN 0-521-62348-0.

[4] K. Lausdahl, P. G. Larsen, and N. Battle, ``A Deterministic Interpreter Simulating A Distributed real time system using VDM,'' in *Proceedings of the 13th international conference on Formal methods and software engineering*, ser. Lecture Notes in Computer Science, vol. 6991. Berlin, Heidelberg: Springer-Verlag, October 2011, pp. 179--194.

[5] T. Kurita and Y. Nakatsugawa, ``The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip,'' *Intl. Journal of Software and Informatics*, vol. 3, no. 2-3, pp. 343--355, October 2009.

[6] T. Oda, Y. Yamomoto, K. Nakakoji, K. Araki, and P. G. Larsen, ``VDM Animation for a Wider Range of Stakeholders,'' in *Proceedings of the 13th Overture Workshop*, June 2015, pp. 18--32, gRACE-TR-2015-06.

[7] 小田朋宏, 荒木啓二郎, ``VDM-SL 実行可能仕様による Web API プロトタイピング環境,'' in **ソフトウェアシンポジウム** *2015* **論文集**, Jun 2015.

[8] T. Oda and K. Araki, ``Overview of VDMPad: An Interactive Tool for Formal Specification with VDM,'' in *International Conference on Advanced Software Engineering and Information Systems (ICASEIS) 2013*, Nov 2013.

[9] T. Oda, K. Araki, and P. G. Larsen, ``VDMPad: a Lightweight IDE for Exploratory VDM-SL Specification,'' in *FormaliSE 2015*. In connection with ICSE 2015, May 2015.

[10] J. Fitzgerald, P. G. Larsen, and S. Sahara, ``VDMTools: Advances in Support for Formal Modeling in VDM,'' *ACM Sigplan Notices*, vol. 43, no. 2, pp. 3--11, February 2008.

[11] N. Battle, ``VDMJ User Guide,'' Fujitsu Services Ltd., UK, Tech. Rep., 2009.

重大不具合に対するOSS開発者の認識調査

松野 祐介
和歌山大学 システム工学部
s171050@sys.wakayama-u.ac.jp

山谷 陽亮
和歌山大学 システム工学研究科
s151049@sys.wakayama-u.ac.jp

大平 雅雄
和歌山大学 システム工学部
masao@sys.wakayama-u.ac.jp

要旨

本研究では，重大不具合 *(High Impact Bug)* に対する *OSS* 開発者の認識を理解することを目的として，*GitHub* に登録されている *OSS* 開発者に対してアンケート調査を行う．大規模・複雑化する近年のソフトウェア開発において，日々報告される多くの不具合に対処することは困難である．そのため，不具合修正プロセスの改善を目的とした研究が盛んに行われているが，個々の不具合の種類については十分に考慮されていない．このような背景から，特に近年，*High Impact Bug (HIB)* が注目されている．*HIB* とはユーザや開発者に重大な影響を与える不具合のことである．しかしながら，*HIB* は主に研究者が着目して研究が進められているものであり，実際には開発者がどのように認識しているのかは十分に明らかではない．そこで本研究では，*GitHub* に登録されている *OSS* 開発者に対して， *HIB* に関するアンケート調査を実施し，その結果を分析した．分析の結果，以下の知見が得た．*(1) OSS* 開発者は，*HIB* の内 *Security* バグを最も重要視している．*(2) OSS* 開発者は，開発スケジュールなど開発に影響を与える不具合よりも，ユーザに影響を与える不具合を *HIB* として認識している．*(3) OSS* 開発者は，ソースコードの複雑さ，不十分なテスト，および，外部環境の変化を *HIB* の主な発生原因と認識している．*(4) OSS* 開発者は，修正パッチのテストや不具合の再現に時間をかけることで *HIB* を修正している．*(5)* 開発者の役割には違いはないが，開発するプロダクトのドメインによって重視する *HIB* は異なる．

1. はじめに

大規模・複雑化する近年のソフトウェア開発では，多くの不具合が報告される．プロジェクトの管理者は，報告された不具合に対して，修正の優先順位の決定，修正コストの見積もり，修正担当者の決定，といった修正プロセスにおける意思決定を行う必要がある．しかし，人員やスケジュールに限りがある中で大量に報告される不具合に迅速かつ適切に対処することは困難である．そのため，不具合修正プロセスの改善を目的として，

- 重複不具合報告の検出 [10, 16]
- 不具合の修正時間予測 [21, 22]
- 不具合箇所の特定 [20, 17]
- 不具合修正タスクの割り当て [1, 23]

などの研究が行われている．

しかし，これらの研究では個々の不具合の種類については十分に考慮されていない．そのため，タイプミスなどの軽微な不具合からセキュリティに関する優先度の高い不具合まで同等に扱われていることが問題となっている．例えば，不具合修正タスクの割り当てを行う際に，個々の不具合に適性のある開発者に割り当てを行う．しかし，重要な不具合である場合に誰が修正を担当すべきであるか考慮されていない．このことから，個々の不具合が与える影響を考慮した分類・分析が必要とされ，近年 High Impact Bug（以降では HIB と呼ぶ）と呼ばれるユーザや開発者に大きな影響を与える不具合に関する研究が行われている [6, 9].

近年の研究では，6 種類の HIB (Surprise バグ，Dormant バグ，Blocking バグ，Security バグ，Performance バグ，Breakage バグ）が注目されている．しかし，先行研究において挙げられている 6 種類の HIB は，いずれも研究者の知識や経験に基づいて定義された不具合であり，開発者が実際に重要な不具合だと認識しているかは明らかではない．そのため，HIB の分析および予測手法の提案を開発者が求めていない可能性がある．

本研究では，オープンソースソフトウェア (OSS) 開発者を対象にアンケート調査を実施し，HIB に対する OSS 開発者の認識を理解することを目的とする．本研究では特に，GitHub に登録されている OSS 開発者を対象としてアンケート調査を実施し，得られた 321 件の回答に対して分析を行う．

以降ではまず，2 章において本研究で対象とする 6 種類の HIB に関する先行研究について述べ，本研究の立場を明らかにする．次に，3 章では，実施するアンケート調査の方法について説明する．4 章では，HIB に対する OSS 開発者の認識理解を目的とするアンケート調査において用いる調査項目の意図について説明する．5 章で分析結果を示し，その結果に踏まえ 6 章で今後必要となる研究の方向性について議論する．最後に 7 章において本論文のまとめと今後の課題を述べる．

2. High Impact Bug

本章では，先行研究で挙げられている 6 種類の HIB の特徴について述べる．その後，本研究のモチベーションを述べる．

2.1. HIB の特徴と先行研究

Surprise バグ [12]

Surprise バグとは，予期せぬ箇所（リリース前にあまり変更されないファイルなど）かつ，予期せぬ時期（特にリリース直後）に発生する不具合を指す．Surprise バグが発生すると，スケジュールの変更が必要になる等，主に開発者に影響を与える不具合として考えられている．Shihab ら [12] は，Surprise バグを予測するモデルを構築する際には，同時変更ファイル数やリリース前の最終変更からの日数等が精度向上に寄与することを示している．

Dormant バグ [2]

Dormant バグとは，あるリリースに向けて開発を行っている際に埋め込まれ，テスト中またはリリース後には見つからずに，次のリリースの後に見つかる不具合を指す．リリース後に報告された不具合は，そのリリースの品質の指標として用いられているが，Dormant バグは，本来不具合が埋め込まれたリリースとは別のリリースの不具合として判断されるため，各リリースの品質評価を誤る可能性がある．そのため，Dormant バグは，他の不具合と比べて影響力が高いため優先して修正が行われる傾向があり，Dormant バグ以外の不具合よりも修正規模は大きいが修正時間は短くなることが Chen らの研究 [2] において報告されている．

Blocking バグ [15]

Blocking バグとは，ソフトウェアコンポーネントの依存関係のために他の不具合の修正を阻害している不具合を指す．Blocking バグが解消されない限り，修正を阻害されている側の不具合が例え軽微なものであっても修正できないため，Blocking バグの存在はプロジェクト全体の不具合修正の長期化を招くとされている．実際，Garcia らの研究 [15] において，Blocking バグは，Blocking バグ以外の不具合に比べて修正時間が長くなることが報告されている．

Security バグ [3]

Security バグとは，それ自体が何かに影響を及ぼすものではなく，攻撃者により悪用される可能性がある不具合を指す．Security バグの代表的な例として，HeartBleed と呼ばれるものが存在する．ソフトウェアに Security バグが含まれていると，攻撃者によってユーザに悪影響が与えられる危険性があるため，影響力の高い不具合として考えられている．Zaman ら [18] は，Security バグは Security バグ以外の不具合よりも修正時間が短いことを報告している．

Performance バグ [8]

Performance バグとは，ユーザーエクスペリエンスやレスポンス，スループットの低下などを含む，プロダクトのパフォーマンス低下を招く不具合を指す．ソフトウェアのパフォーマンスはユーザの満足度に影響を与える非機能要求として重要視され，プロジェクトの成功に大きく関わると考えられている．明確な障害が発生する不具合から，ユーザに

よって認識が分かれる軽微なパフォーマンス低下まで，様々な不具合が Performance バグとして扱われる．不具合報告のテキスト情報に含まれるキーワード（perf, slow, hang, throughput など）を用いて Performance バグを分類する研究 [19, 8] などがある．

Breakage バグ [12]
Breakage バグとは，不具合の修正や新しい機能の追加によって以前使えていた機能を使えない状態にする不具合を指す．Breakage バグが発生すると，既存の機能が失われることで，ユーザが既存の機能で行っていた日常業務に支障をきたすため，主にユーザに影響を与える不具合として考えられている．Shihab ら [12] は，Breakage バグの予測にはファイル変更回数やリリース前のバグ数が重要であると報告している．

2.2. 本研究のモチベーション

先行研究で取り上げられている HIB はすべて，研究者の知識や経験に基づいて問題視されている不具合であり，必ずしも実務者の認識と一致しているとは限らない．研究者が提案する HIB 検出のための予測モデルや分析手法が実際に役立つかどうかが不明なままであり，実務者のニーズを正しく理解しないままさらに研究を行うことは好ましくない．

そこで本研究では，実務者の HIB に対する認識をインタビュー調査することにより，実務者の観点から今後の研究の方向性や方針についての知見を得ることを目指す．ただし，先行研究のほとんどは，OSS プロジェクトから収集した不具合データを用いて予測モデルの検証結果や分析結果を示している．本研究においても，結果の一般性を高めることを目的として様々な OSS プロジェクトの開発者を対象として HIB に対する認識を調査する．企業実務者のニーズとは依然として乖離が生じる可能性もあるが，本研究で得られる知見を今後の議論の叩き台として活用したいと考えている．

3. 調査方法

本章では，実施するアンケート調査の対象と手順について説明する．

3.1. 調査対象

本研究では，OSS における開発者の HIB に対する認識を調査するために，GitHub[1] に登録されている開発者を調査対象とする．GitHub とはソフトウェア開発プロジェクトのためのホスティングサービスであり，開発者は Git によって管理されるリポジトリを GitHub 上で公開することにより，世界中の開発者とソースコードなどを共有することができる．ユーザ数 1,000 万，リポジトリ数 3,500 万を超える最も大規模なホスティングサービスであるため，多様な種類の OSS の開発に携わる開発者から HIB に対する認識を調査できると考えた．

3.2. 調査手順

3.2.1. 開発者の選定

本研究では，GitHub に登録されている開発者の中から活動的な開発者に対してアンケート調査を実施する．活動量の指標として GitHub API[2] を用いて取得できる contribution（リポジトリに対してコミットした回数）を用いるが，GitHub ユーザ個々人の contribution を直接カウントする方法はないため，以下の手順により contribution の多い開発者を選定した．

1. リポジトリ情報の取得
   GitHub API を用いてリポジトリ一覧を取得する．リポジトリは 3,500 万件以上存在するが，今回の調査では 3,500 万件までを用いる．リポジトリ一覧に基づいて再び GitHub API を用いて各リポジトリから contribution の多い順に 30 名のユーザ名（開発者のアカウント名）を取得する．

2. リポジトリの選定
   各リポジトリの contributor を参照し，全員（最大 30 名）の contribution を合計した値を各リポジトリの contribution とする．contribution が多い順にソートしリポジトリのランキングを作成する [3]．ランク付けしたリポジトリの内，contribution が 1,000

[1] GitHub:https://github.com/
[2] https://api.github.com/
[3] フォークにより作成されたリポジトリにはフォーク元リポジトリの contributor のデータもコピーするため，重複してデータ取得してしまう問題が発生する．この問題を回避するために，オーナーが異なる複数の同名リポジトリが存在する場合には，contoribution の一番多いリポジトリのみを残し，それ以外をランキングから除外した．

回以上のリポジトリ 1,211,913 件を活発なリポジトリとして選定した．

3. 開発者情報の取得
   GitHub API を用いて，手順 2 で選定したリポジトリに登録されている開発者のメールアドレスを取得する．ここで，開発者のアカウント名と contribution およびメールアドレスを紐付けておく．

4. 開発者の選定
   手順 3 の処理を行うことで，異なるリポジトリで活動している同じ開発者のデータが複数得られる．同じ開発者のデータが複数あれば contribution を合計する．contribution の多い順にソートし開発者のランキングを作成する．メールアドレスが無記入だった開発者を除いた 54,282 人の内，contribution が 100 回以上の開発者 22,228 人をアンケート対象として選定した．

3.3 アンケートの実施

前述の手順によって選定した 22,228 人の開発者の内，メールサーバの制限により，18,299 人にメールでアンケート調査への参加を依頼した．その結果，321 人からの回答が得た．アンケート調査は 2015 年 7 月から 2015 年 8 月の間に行った．アンケートには，開発者のプロファイルに関する情報（開発経験，開発目的，プロジェクトでの役割，主要参加プロジェクトなど）の他，次章で説明する調査項目が含まれる．回答方法は，一部の調査項目を除いて，著者らが用意した選択肢から 1 つ選ぶ形式のものであり，適切な選択肢がない場合には「その他 (other)」を選択した上で自由形式でその内容を記述できるようにした．

4. 調査内容

本章では，開発者の HIB に対する認識を理解するために行う調査の内容について説明する．

4.1. High Impact Bug の重要性

Q1：6 種類の HIB の中で最も重要なものはどれか？
調査の意図：近年研究者が注目している 6 種類の HIB の内どの HIB が，OSS の開発現場で重要な不具合として認識されているかは明らかではない．Q1 は，OSS 開発者が最も重要視する HIB を特定するための調査項目である．

Q2：6 種類の HIB 以外の HIB は存在するか？
調査の意図：本研究が対象とする 6 種類の HIB は，あくまでも研究者が研究対象として近年取り上げているものであり，OSS の開発現場では 6 種類の HIB 以外にも HIB として認識されている不具合が存在する可能性がある．Q2 は，6 種類の HIB 以外に重要視する HIB を特定するための調査項目である．

4.2. High Impact Bug の影響範囲

Q3：HIB は誰に対して最も影響があるか？
調査の意図：先行研究が 6 種類の HIB に着目している理由は，それらがユーザあるいは開発者に対して大きな影響を与えるためである．例えば，Surprise バグは予期しないタイミングで予期しない箇所で発見されるため，開発スケジュールの再調整に至るケースがあり，開発者に大きな影響を与えるとされている．Breakage バグは以前使えていた機能を使えなくする不具合であるため，ユーザの業務や利用体験に大きな影響を与えるとされている．直接的，間接的にはいずれの HIB もユーザおよび開発者の双方に影響を与えるものではあるものの，Q3 は，OSS 開発者が各 HIB の影響範囲をどのように捉えているかをより正確に把握するための調査項目である．

4.3. High Impact Bug の発生原因・対処方法

Q4：HIB の発生原因はどのようなものがあるか？
調査の意図：先行研究では HIB の発生原因に対する分析が行われている．例えば，Dormant バグの発生原因は主に，境界値や制御フローであることが報告されている [2]．Q4 は，各 HIB の発生原因に対する OSS 開発者の認識を理解するための調査項目である．

Q5：HIB をどのように修正するか？
調査の意図：先行研究では HIB の修正方法に関する分析が行われている．例えば，Security バグは多くの開発者が修正活動に関与し，特に熟練の開発者に担当されやすいことが報告されている [18]．Q5 は，OSS 開発者が HIB をどのように修正しているのかを理解するための調査項目である．

4.4. 役割とドメインによる認識の違い

Q6：プロジェクト内の役割によって HIB に対する認識の違いがあるか？
調査の意図：OSS 開発では，開発者にはプロジェクト全体の管理，ソフトウェアのテスト，不具合報告，修正パッチの投稿・レビュー・反映といった様々な役割が存在し，プロジェクト内で担う役割によって HIB に対する認識に違いが存在する可能性がある．例えば，不具合修正を担当する開発者は，他の不具合修正を妨害する Blocking バグを最も重要視するかもしれない．Q6 は，OSS 開発者の役割の違いによる HIB に対する認識の違いを明らかにするための調査項目である．
Q7：開発するプロダクトのドメインによって HIB に対する認識の違いがあるか？
調査の意図：OSS 開発では，ビジネス向けアプリケーション（OpenOffice など）から開発者向けアプリケーション（Eclipse IDE など），ゲームアプリケーションまで様々なドメインのプロダクトが開発されている．OSS 開発者がどのドメインのプロダクトを開発しているかによって，HIB に対する認識に違いが存在する可能性がある．例えば，ブラウザなどの一般向けに広く普及しているプロダクトの開発者は，Performance バグを重要視するかもしれない．Q7 は，開発するプロダクトのドメインの違いによる HIB に対する認識の違いを明らかにするための調査項目である．

5. 調査結果

本章では，アンケート調査の結果を示す．

5.1. High Impact Bug の重要性

Q1：6 種類の HIB の中で最も重要なものはどれか？
回答結果を図 1 に示す．縦軸は回答の割合を，横軸は 6 種類の HIB およびその他 (other) を表す．6 種類の HIB の内，Security バグが 53.1%と最も高く，次点で Breakage バグが 22.4%だった．Surprise バグ，Dormant バグ，Blocking バグ，Performance バグに関しては回答の割合は 2〜6%であった．
Q2：6 種類の HIB 以外の HIB は存在するか？
Q1 においてその他 (other) を選択した開発者が 7.5%存在した．other を選択した場合には自由記述ができるようにしていたため，いくつか具体的な回答が得られた．

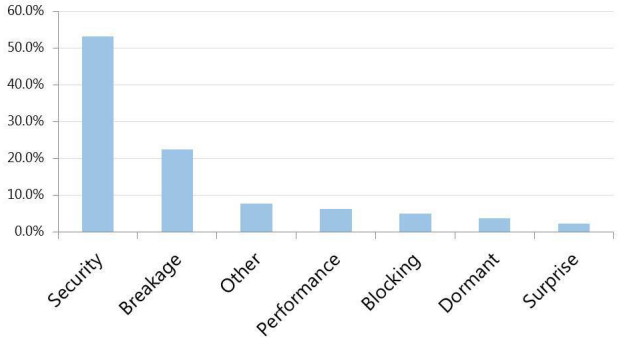


図 1. Q1 の調査結果（other はその他を選択した割合）

自由記述欄にコメントがあった回答の回答番号と回答内容を表 1 に示す．表 1 より，6 種類の HIB と類似する不具合も含まれていたが，ユーザデータの消失やユーザ体験の低下など，ユーザに対して大きな影響を与える不具合を HIB としている開発者が多く見受けられた．

5.2. High Impact Bug の影響範囲

Q3：HIB は誰に対して最も影響があるか？
回答結果を図 2 に示す．縦軸は回答の割合を表す．横軸は回答者全員 (all) と各 HIB を選択した開発者を表す．Blocking バグ以外の HIB に対しては，開発者の過半数が「主にユーザに影響を与える」不具合として認識していることが分かる．特に Performance バグについては，95%の開発者がユーザに影響を与える不具合と捉えている．一方，Blocking バグについては，50.0%の開発者が「主に開発者に影響を与える」不具合と考えており，先行研究が示した Blocking バグの問題意識については OSS 開発者と認識を共有しているものと言える．Surprise バグおよび Dormant バグについては，先行研究の知見とは異なり，ユーザ視点からその影響を考えている開発者が大半を占めることが分かった．

5.3. High Impact Bug の発生原因・対処方法

Q4：HIB の発生原因はどのようなものがあるか？
回答結果を図 3 に示す．縦軸，横軸の意味は Q3 と同様である．回答全体としては，「コードの複雑さ」が 32.7%，「不十分なテスト」が 30.5%，「外部環境の変化」が 18.9%と

表 1. Q1 における other の自由既述（Q2 の調査結果）

回答番号	回答内容
#13	A bug resulting in false results
#28	threatening users' data security
#32	a bug impacting user experience
#38	bug affecting many users
#72	a mix between security/performance/core bugs
#114	lost user data
#122	a bug reducing user satisfaction
#128	bug having most user impact (as this is judged critical by management)
#157	bug corrupting the database silently
#164	bug in core functionality that affects only particular platform and cannot be easily worked around
#169	a bug threatening systems features
#187	calculation error
#195	bug blocking main functions
#240	causing data loss for users
#276	a bug preventing usage of the project, e.g. crashes
#309	Concurrency bugs

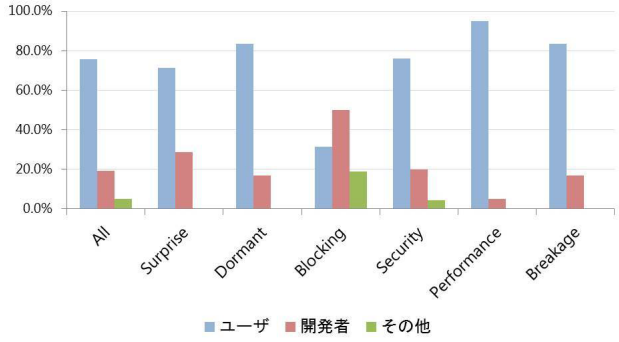


図 2. Q3 の調査結果

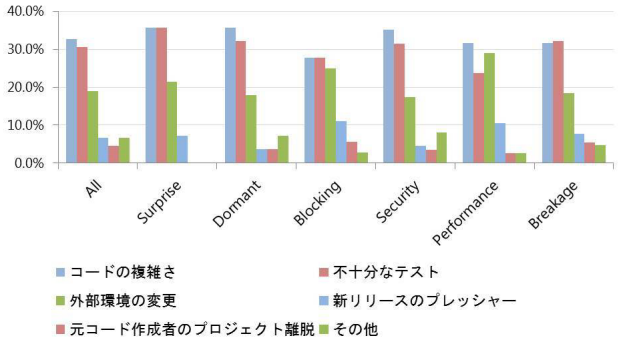


図 3. Q4 の調査結果

いう結果となり，HIB の主な発生原因であることが分かった．HIB 毎の結果も同様の傾向があるが，特に Performance バグが最も重要とした開発者の 28.9%は「外部環境の変化」を主要な発生原因と回答しており，他の HIB よりも高い割合となっていた．OS のアップデートや計算機環境の変更により，OSS においてもパフォーマンスに関する問題が発生しやすいことが伺える．

Q5：HIB をどのように修正するか？

回答結果を図 4 に示す．縦軸，横軸の意味は Q3 と同様である．回答全体としては,「修正パッチのテストを慎重に行う」が 32.5%,「不具合の再現に時間をかける」が 24.5%,「熟練の開発者に修正を割り当てる」が 20.7%,「十分に議論を行う」が 16.8%,「その他」が 5.5%の順となった．HIB 毎の結果も概ね同様の傾向があるが，特に Surprise バグおよび Performance バグが最も重要とした開発者は,「不具合の再現に時間をかける」を最も多く選択している（Surprise: 37.5%, Performance: 35.0%）．Surprise バグは予期しない箇所やタイミングで発見されることや，Performance バグは絶対的な判断基準が存在しないことから，不具合を再現し，開発者同士で問題を共有することが重要になるため，他の HIB よりも不具合の再現を重要視していると推察される．

5.4. 役割とドメインによる認識の違い

Q6：プロジェクト内の役割によって HIB に対する認識の違いがあるか？

回答結果を図 5 に示す．縦軸は回答の割合を，横軸は回答者全員（全体）と開発者の役割を表す．全体の結果を含め，全ての役割において Security バグが最も重要視されていることが分かる．次点では，Breakage バグを選

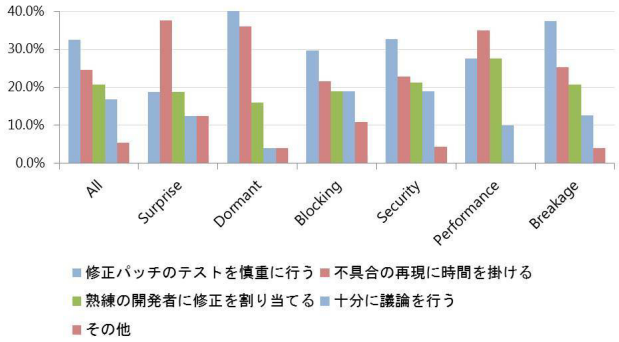


図 4. Q5 の調査結果

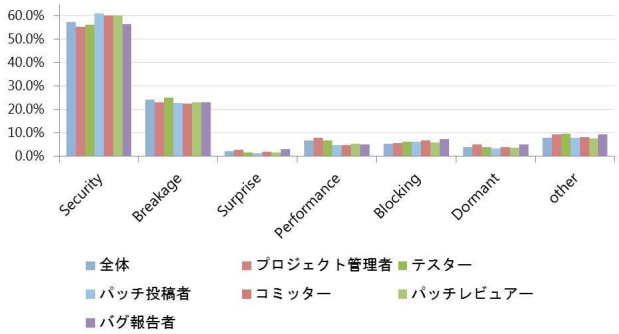


図 5. Q6 の調査結果

択する開発者の割合が高い結果となっている．このことから，OSS 開発者の役割の違いによって重要視する HIB に大きな違いは見られず，開発者間で一致した見解が存在することが分かった．

Q7：開発するプロダクトのドメインによって HIB に対する認識の違いがあるか？

Q7 は開発するプロダクトの種類（ドメイン）によって HIB に対する認識の違いが存在するかを調査するためのものであるが，GitHub には階層型のドメイン分類が存在しないため，本研究では，SourceForge[4] の分類を参考に，本調査で回答した開発者が開発している OSS を目視で確認し分類した．参考にしたカテゴリは，Audio & Video，Business & Enterprise，Communications，Development，Home & Education，Games，Graphics，Science & Engineering，Security & Utilities，System Administration の 10 種である．図 6 に分類結果を示す．縦軸は回答の割合を，横軸は回答の割合を，縦軸は全

[4]Source Forge:http://sourceforge.net/

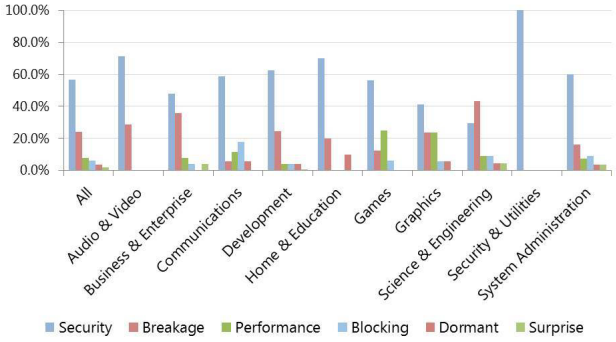


図 6. Q7 の調査結果

カテゴリ (all) と各カテゴリを表す．全体の結果としては，Security バグが 56.7%，Breakage バグが 24.1%と，他の HIB よりも突出して割合が高いことが見て取れる．ドメイン別に見ても概ね全体の結果と一致する傾向にあるが，Communications に関しては Security バグについで重要視されるのが Blocking バグであり，Science & Engineering では最も重要視されるのが Breakage バグとなっている．また，Games や Graphics においては，Security バグに続いて Performance バグが次点として選ばれている．なお，Security & Utilities では，(当然ではあるかもしれないが) Security バグが 100.0%となっている．これらの結果から，開発しているプロダクトのドメインによって重要視する HIB に特徴があることが分かった．

6. 考察

本章では，本研究で行ったアンケート調査の結果を基に考察を行う．

6.1. High Impact Bug の重要性

Q1 の結果，半数以上の OSS 開発者が Security バグを HIB だと認識していることが分かった．これは Security バグが攻撃者に悪用されることによってユーザに損害を与える可能性があるため，多くの開発者にとって優先して修正すべきだと認識されていると考えられる．また，次点で 22.4%の OSS 開発者が Breakage バグを HIB だと認識していることが分かった．これも同様に既存の機能が失われることによってユーザの日常業務が滞り，損害を与える可能性があるためと考えられる．

Q2 の結果からも同様の認識が示唆される．表 1 が示すとおり，#28 の OSS 開発者はユーザデータのセキュリティを脅かす不具合を HIB だと認識している．その他にも，#72 はセキュリティに関する問題を含む複合的な不具合，#114 および#240 はユーザデータの損失を引き起こす不具合が HIB として挙げている．

このことから，Security バグや Breakage バグを含む，ユーザに損害を与える可能性のある不具合に着目した研究が今後も重要であることが分かる．例えば，報告される不具合が Security バグ [3] であるかの予測や，Breakage バグの特定 [12] といった手法をさらに洗練させることが OSS 開発者にとっては特に有益であると思われる．

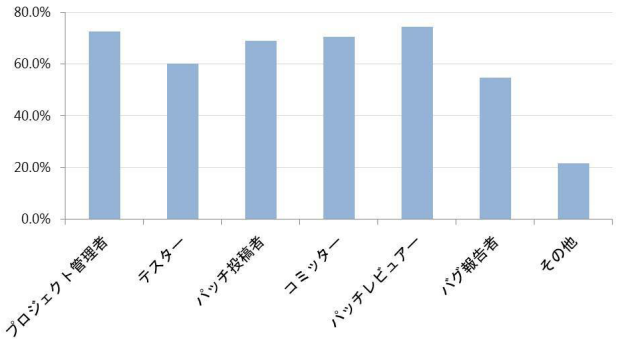


図 7. アンケートに回答した OSS 開発者の役割（複数選択あり）

6.2. High Impact Bug の影響範囲

Q3 の結果，Blocking バグを除いて，開発者の大半が HIB はユーザに大きな影響を与えるものであると認識していることが分かった．特に Performance バグを HIB であると回答した開発者の 95.0%がユーザに影響を与える不具合と考えており，ユーザ視点で HIB を重視する開発者には，前述の Security バグおよび Breakage バグに関する研究に加え，Performance バグに関する研究が求められていると言える．[19, 8] では，Performance バグの特徴や修正方法などについて分析されているのみであり，今後は Performance バグの再現や原因特定を支援する研究が必要である．

6.3. High Impact Bug の発生原因・対処方法

Q4 の結果，HIB は「ソースコードの複雑さ」,「不十分なテスト」,「外部環境の変化」が主な発生原因と捉えられており，どの HIB においても概ね同様の傾向があることが分かった．Eclipse プロジェクトなどの大規模 OSS 開発プロジェクト以外の OSS プロジェクトでは，厳格なリリース計画やマイルストーンを設定していることは少ない．また，特に GitHub では，プルリクエストベースの開発が行われているため，コアメンバー以外の開発者（貢献者）からのコードやパッチ提供を受けつつ，実装と不具合修正が同時進行する形で開発が行われる．小規模あるいは中規模プロジェクトでは人的資源にも限界があるため，リファクタリングやテストを十分行わずに（行えずに）リリースする状況が多く発生するのではないかと考えられる．近年では，HeartBleed 問題で注目されたように，少人数で開発されている OSS が社会的に広く普及していることも少なくない．技術的な支援や研究も必要であるが，人的な支援も今後はより重要になると考える．

Q5 の結果，特に Security バグおよび Performance バグの修正方法として,「不具合の再現に時間をかける」と回答した開発者の割合が高いことが分かった．これらの不具合は，不具合の再現が困難なだけではなく修正も困難を伴う場合が多く，熟練の開発者に不具合の修正を割り当てることが多いと指摘されている [18]．不具合の再現方法や特定方法に加え，バグトリアージ技術 [1, 23] の向上により，効率的に適任の開発者に不具合修正タスクを依頼できるようにすることが重要であると思われる．

6.4. 役割とドメインによる認識の違い

Q6 の結果，OSS 開発者の役割によって HIB に対する認識に大きな差は見られなかった．これは本研究でアンケートに回答した OSS 開発者の役割に明確な差がなかったことが一因と考えられる．図 7 に，本調査に参加した OSS 開発者の役割の分布を示す．プロファイルの調査では，複数選択を可能としたため多くの開発者が複数の役割を選択していた．このことが，役割間の認識の差を小さくした原因と思われる．今後は，各役割毎に HIB の認識を調査できるようにする（「テスターとして重要視する HIB はどれか？」などの質問形式にする）など工夫が必要である．

Q7 の結果，開発するプロダクトのドメインによって HIB に対する認識に大きな差が見られた．ドメインの認識を考慮することで，プロジェクトに則した効果的な支

援手法の構築が期待できる．例えば，バグトリアージ技術やバグローカリゼーション技術 [11, 17] を構築する際には，プロジェクトが重要視する HIB に重みを置いた手法を用いることで，より効果的な支援が行える可能性がある．

6.5. 制約

本研究では，GitHub に登録されている OSS 開発者を対象にアンケート調査を実施した．そのため，GitHub で活動していない OSS 開発者や企業の開発者は調査対象から外れている．また，GitHub に登録されている開発者の中で活動的な開発者に対してアンケート調査を実施するために，活動量の指標として GitHub API で取得できる contribution を用いた．この contribution にはコミット数のみが反映され，それ以外の issue 開設，コメント投稿といった活動が反映されていない．そのため，コミット以外の活動で多く貢献している OSS 開発者が調査対象から外れている．また，活動的な開発者として選定した中で，メールアドレスを GitHub 上で公開していない OSS 開発者は調査対象から外れている．その他，メールサーバの制限により，対象者の全員にメール送信ができなかった．一般性の高い結果を得るためには，本研究で対象にできなかった開発者に対しても調査を行う必要がある．

なお，本研究ではアンケート調査の回答を多く得るために質問数が多くならないようにしたが，開発者の認識をより良く理解するためには，今後さらに多くの質問を実施する必要がある．また，回答のほとんどは複数の選択肢から 1 つだけを選択する単一回答方式を用いた．そのため，OSS 開発者が重要視する HIB に対して回答の偏りが生じている可能性は否定できない．今後は，回答形式に順位回答を用いることで懸念を解消できる可能性がある．

7. まとめと今後の課題

本研究では，OSS 開発者の HIB に対する認識を理解するために，GitHub に登録されている OSS 開発者を対象にアンケート調査を実施した．得られた 321 件の回答を分析した結果，主に以下の知見が得られた．

- OSS 開発者は，HIB の内 Security バグを最も重要視している．
- OSS 開発者は，開発スケジュールなど開発に影響を与える不具合よりも，ユーザに影響を与える不具合を HIB として認識している．
- OSS 開発者は，ソースコードの複雑さ，不十分なテスト，および，外部環境の変化を HIB の主な発生原因と認識している．
- OSS 開発者は，修正パッチのテストや不具合の再現に時間をかけることで HIB を修正している．
- 開発者の役割には違いはないが，開発するプロダクトのドメインによって重視する HIB は異なる．

また，アンケート調査の結果から，不具合の特定や再現，割り当て方法に関して HIB を考慮した研究が今後必要となることが分かった．今後は，アンケート対象を拡充するとともに調査項目を充実させて調査を継続したい．また，企業実務者との意見交換などを踏まえて，不具合修正プロセスの改善技術の方向性についても有益な知見を提供したいと考えている．

謝辞

本研究の一部は，文部科学省科学研究補助金（基盤 (C): 15K00101）による助成を受けた．

参考文献

[1] John Anvik, Lyndon Hiew, and Gail. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*, pp. 361–370, 2006.

[2] Tse-Hsun Chen, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. An empirical study of dormant bugs. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 82–91, 2014.

[3] Michael Gegick, Pete Rotella, and Tao Xie. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th Working Conference on Mining Software Repositories (MSR '10)*, pp. 11–20, 2010.

[4] Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE '10)*, pp. 52–56, 2010.

[5] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs.

In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*, pp. 111–120, 2009.

[6] Yutaro Kashiwa, Hayato Yoshiyuki, Yusuke Kukita, and Masao Ohira. A pilot study of diversity in high impact bugs. In *Proceedings of 30th International Conference on Software Maintenance and Evolution (ICSME2014)*, pp. 536–540, 1 2014.

[7] Lionel Marks, Ying Zou, and Ahmed E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE '11)*, pp. 11:1–11:8, 2011.

[8] Adrian Nistor, Tian Jiang, and Lin Tan. Discovering, reporting, and fixing performance bugs. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, pp. 237–246, 2013.

[9] Masao Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, Keisuke Fujino, Hideaki Hata, Akinori Ihara, and Kenichi Matsumoto. A dataset of high impact bugs: Manually-classified issue reports. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*, pp. 518–521, 2015.

[10] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*, pp. 499–510, 2007.

[11] Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E. Perry. Improving bug localization using structured information retrieval. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE'13)*, pp. 345–355, 2013.

[12] Emad Shihab, Audris Mockus, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. High-impact defects: A study of breakage and surprise defects. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pp. 300–310, 2011.

[13] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10)*, pp. 45–54, 2010.

[14] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pp. 365–375, 2011.

[15] Harold Valdivia Garcia and Emad Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 72–81, 2014.

[16] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 461–470, 2008.

[17] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In *Proceedings of 30th International Conference on Software Maintenance and Evolution (ICSME2014)*, pp. 181–190, 2014.

[18] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. Security versus performance bugs: A case study on firefox. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*, pp. 93–102, 2011.

[19] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. A qualitative study on performance bugs. In *Proceedings of the 9th Working Conference on Mining Software Repositories (MSR '12)*, pp. 199–208, 2012.

[20] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 14–24, 2012.

[21] 正木仁, 大平雅雄, 伊原彰紀, 松本健一. Oss 開発における不具合割当パターンに着目した不具合修正時間の予測. 情報処理学会論文誌, Vol. 52, No. 2, pp. 933–944, 2 2013.

[22] 吉行勇人, 大平雅雄, 戸田航史. Oss 開発における管理者と修正者の社会的関係を考慮した不具合修正時間予測. コンピュータソフトウェア, Vol. 32, No. 2, pp. 128–134, 6 2015.

[23] 柏祐太郎, 大平雅雄, 阿萬裕久, 亀井靖高. 大規模 oss 開発における不具合修正時間の短縮化を目的としたバグトリアージ手法. 情報処理学会論文誌, Vol. 56, No. 2, pp. 669–681, 2 2015.

製品開発におけるOSS導入のための OSS事前評価手法確立に向けた調査

松本 卓大[†] 山下 一寛[†] 亀井 靖高[‡] 鵜林 尚靖[‡]
九州大学大学院システム情報科学府
[†]{matsumoto, yamashita}@posl.ait.kyushu-u.ac.jp
[‡]{kamei, ubayashi}@ait.kyushu-u.ac.jp

大浦 雄太 岩崎 孝司 高山 修一
富士通九州ネットワークテクノロジーズ
(QNET)

要旨

本研究では，製品にオープンソースソフトウェア（*OSS*）を導入するか否かを判断する目安として，*OSS*の品質を事前に把握するための評価手法の確立を目指す．本発表では，事前評価を開発の現場で実施するにあたって必要となる *1*）開発者が *OSS* に求める品質，*2*）品質を評価するために必要な評価指標の取得時間の *2* 点について調査を行った．

1. 製品開発におけるOSS導入と課題

近年，開発コストの削減，および，高品質な製品開発実現のために，企業における製品開発へのOSS導入が増えている [1]．しかしながら，低品質なOSSを適用することで製品自体の品質を低下させてしまい，その品質改善のために見積もり以上のコストがかかってしまうことがある．製品にOSSを導入することによる想定外のコスト増加を防ぐためには，導入するOSSの品質を事前に把握しておくことが望ましい．

OSSの事前評価に関して，これまでにも様々な取り組みが行われてきた [2][3]．しかしながら，既存のOSSの事前評価に関する取り組みは,「取得方法が不明な評価指標の存在」,「スコアリングルールが曖昧な項目が半数」などの問題が挙げられる．いずれの事前評価手法も開発の現場での利用にまで定着していない．事前評価を実際に開発の現場で実施するためには，実施にあたって障害となっている問題に関して調査する必要がある．

本稿では，OSSの品質を事前に評価するための手法を確立することを目的とし，事前評価を実際に開発の現場で実施するにあたって必要となる項目に関して行った調査項目について述べる（詳細はスライド参照）．

2. 調査項目

開発者がOSSに求める品質：開発プロジェクトによってOSSの利用形態は異なり求めるOSSの品質も異なってくる．各プロジェクトが求めるOSSの品質について，QNETの開発部門を対象にヒアリングでの調査を行った．

品質を評価するために必要な評価指標の取得時間：事前評価にあたって評価指標が必要となる．継続的に事前評価を行うにはOSSの進化にあわせて評価指標を更新する必要があり費用がかかる．更新のための費用が大きすぎると実用的ではない．必要な評価指標に対して手動で取得を行い，OSS1件あたりの取得時間について調査を行った．

参考文献

[1] 独立行政法人情報処理推進機構. 第3回オープンソースソフトウェア活用ビジネス実態調査. 2010.

[2] Etiel Petrinja, Alberto Sillitti, and Giancarlo Succi. Com- paring openbrr, qsos, and omm assessment models. In Open Source Software: New Horizons, pp. 224238. Springer, 2010.

[3] OSS Northeast Asia. Reposs: A flexible oss assessment repository. 2012.

ISBSG データを用いた見積もり研究に対する IPA/SEC データを用いた追試

山田悠斗
大阪大学　大学院情報科学研究科
y-yuto@ist.osaka-u.ac.jp

江川翔太
大阪大学　大学院情報科学研究科
s-egawa@ist.osaka-u.ac.jp

角田雅照
近畿大学　理工学部情報学科
tsunoda@info.kindai.ac.jp

楠本真二
大阪大学　大学院情報科学研究科
kusumoto@ist.osaka-u.ac.jp

要旨

ソフトウェア開発プロジェクトの見積もり技術に関する研究が盛んに行われている．見積もりに関する研究の評価には *ISBSG* データが用いられることが多い．一方，実証的ソフトウェア工学の分野では既存研究で得られた知見に対する追試 *(replication study)* が重要であるとされている．追試を行うことによって特定の条件や環境において得られた知見に対する再現性や，異なる実験条件から得られる知見の差異を調査することができる．また，追試を行い様々な条件から得られた知見を統合することによって，新規開発プロジェクトにおいてそれと類似した条件での知見を再利用することが可能となる．

本稿では，*ISBSG* データが用いられている見積もりの既存研究を対象にした追試を実施する．既存研究とは異なるデータセットを用いて実験を行うことで，得られる知見に差異が生じるかを調査する．過去 *5* 年間で *ISBSG* データが利用されていた論文 *22* 本の中から *4* 本の論文を選択し，*IPA/SEC* データを用いて追試を行った．その結果，*2* 本の論文では既存研究と類似した知見が，残りの *2* 本の論文では既存研究と異なる知見が得られた．

1. はじめに

ソフトウェア工学はソフトウェアの開発，運用，保守に関して体系的，定量的にその応用を考察する分野であり，この分野で扱われている技術の中にソフトウェア開発の見積もり [1] がある．ソフトウェア開発においては初期の段階から全体のコストや工数を正確に見積もることが重要であるとされており [2]，見積もりの誤りがプロジェクト失敗へと繋がる場合がある．この問題を解決するためにソフトウェア開発の見積もりに関する研究が盛んに行われている．見積もりに関する研究の評価には ISBSG データ [3] が用いられることが多い．過去 5 年間における主要国際会議の論文誌を調査した結果では，見積もりに関する研究が行われている 83 本の論文のうち 22 本の論文で ISBSG データが利用されていた．

また，実証的ソフトウェア工学 [4] の分野では，既存研究で得られた知見に対する追試 (replication study) が重要であるとされている [5, 6]．研究に対する追試とは，ある研究に関して，実験の条件や環境を部分的に変更して実験を再現することである．追試を行うことによって特定の条件において得られた知見が別の条件においても再現できるか，異なる条件では別の知見が得られるか等を調査することができる．これらを調査することで，研究成果に対する妥当性の評価を行うことができる．また，様々な条件から得られた知見を統合することで，新規開発プロジェクトにおいてそれと類似した条件での知見を再利用することが可能となる．

そこで，本研究では ISBSG データが用いられている見積もりの既存研究を対象にした追試を実施する．既存研究とは異なるデータセットで実験を行うことで，得られる知見に差異が生じるかを調査する．異なる知見が得られた場合は結果が異なった原因の考察や追加の調査に繋げることができる．同様の知見が得られた場合は既存研究の研究成果の外的妥当性がより高められたと判断することができる．

過去 5 年間で ISBSG データが利用されていた論文の中から 4 本の論文 [7, 8, 9, 10] を選択し，IPA/SEC データを用いて追試を行った．その結果，4 本中 2 本の論文では既存研究と類似した知見が得られたが，他の 2 本の論文では既存研究と一部異なる知見が得られた．

以降，2. では研究の背景となる関連研究について述べる．3. では追試を行うための準備事項について述べる．4. では選択した論文に対して追試を行った結果と考察について述べる．5. では主な結果と今後の課題をまとめる．

2. 準備

本章では研究の背景となる諸用語や関連研究について簡単に述べる．

2.1. 線形重回帰分析

ソフトウェア開発規模の見積もりには，多変量回帰分析の一手法である線形回帰分析が多く用いられている [11]．線形回帰分析では予測対象となる目的変数と予測に必要とする説明変数との関係を一次式で表したモデルが作成される．一般的なモデルは式 (1) の形で表される．目的変数である $\hat{Y}$ には予測の対象となる工数等が当てはめられ，説明変数である X_i にはソフトウェアの規模や要因といった予測対象を導くために必要となる要素が当てはめられる．予測対象の実測値と予測値の残差が最小となるように a_i と b が決められる．

$$\hat{Y} = a_1X_1 + a_2X_2 + \cdots + a_nX_n + b \tag{1}$$

精度の推定には以下の式 (2)〜(6) によって求められる 5 つの評価指標 [12] である AR (Absolute Residuals)，MRE (Magnitude of Relative Error)，MER (Magnitude of Error Relative to the estimate)，BRE (Balanced Relative Error)，Pred(25) が多く用いられる．MRE は実測値から見た予測値の相対誤差を，MER は予測値から見た実測値の相対誤差を表す．BRE は過大見積もりや過小見積もりに対しバランスの良い評価を行うことができる．AR，MRE，MER，BRE は値が小さいほど，Pred(25) は値が大きいほど見積もり精度が良いと評価される．また，例えば MRE の平均値を MMRE などと表し，中央値を MdMRE などと表す．

$$\text{AR} = |\ \text{実測値} - \text{予測値}\ | \tag{2}$$

$$\text{MRE} = \frac{\text{AR}}{\text{実測値}} \tag{3}$$

$$\text{MER} = \frac{\text{AR}}{\text{予測値}} \tag{4}$$

$$\text{BRE} = \begin{cases} \text{MRE} & (\text{予測値} - \text{実測値} \geq 0) \\ \text{MER} & (\text{予測値} - \text{実測値} < 0) \end{cases} \tag{5}$$

$$\text{Pred(25)} = \frac{\text{評価指標の値が 0.25 以下であるデータ数}}{\text{全データ数}} \tag{6}$$

回帰モデルの予測精度を表す指標として，他に決定係数がある．これは重相関係数の 2 乗に等しく，説明変数が目的変数をどの程度説明できるかを表す．この値が大きいほど説明変数と目的変数の相関が強く，得られたモデルの予測能力が高いことを意味する．

2.2. ファンクションポイント法

ソフトウェアの規模を見積もる手法の 1 つにファンクションポイント法 [13] がある．この手法では，まずソフトウェアの持つ機能から 5 種類の基本機能要素を抽出し，それぞれの処理内容の複雑度からファンクションポイント (以降，FP) と呼ばれる点数を付ける．この FP から工数等の推定が行われる．5 種類の基本機能要素とは以下に示す要素のことを言う [14]．

内部論理ファイル (ILF)
: 計測対象のアプリケーション内でデータが更新される論理的な関連を持ったデータの集合

外部インターフェイスファイル (EIF)
: 計測対象のアプリケーションによってデータが参照されるデータの集合 (データは更新されない)

外部入力 (EI)
: 計測境界外からのデータ入力によって ILF の更新を行う処理

外部出力 (EO)
: 計測境界外へのデータ出力を含む処理のうち，出力データに派生データを含むもの

外部参照 (EQ)
: 計測境界外へのデータ出力を含む処理のうち，出力データに派生データを含まないものであり，処理が ILF を更新しないもの

特に，アプリケーション全体での FP の合計をアプリケーション FP と言い，アプリケーション FP にシステムの特性を考慮に入れて調整を加えた値を調整済みアプリケーション FP と言う．

2.3. ISBSG データ

ISBSG データとは，ISBSG (The International Software Benchmarking Standards Group) [3] が世界 24 ヶ国に存在する組織や企業から実開発のデータを収集し，整理したデータセットである．開発工数やソフトウェアの規模，開発言語等のデータが収録されている．リリースごとにデータ数は異なるが，最新のデータセットには 5000 以上のプロジェクトデータが 118 項目に分けて蓄積されている．

見積もりに関する研究の評価にはこの ISBSG データが用いられることが多い．過去 5 年間における主要国際会議の論文誌を調査した結果では，見積もりに関する研究が行われている 83 本の論文のうち 22 本の論文で ISBSG データが使用されていた．

2.4. IPA/SEC データ

IPA/SEC データとは，独立行政法人情報処理推進機構 [15] が日本に存在する組織や企業から実開発のデータを収集し，整理したデータセットである．2014-2015 版では，3541 プロジェクトのデータが 194 項目に分けて蓄積されている．

2.5. 実証的ソフトウェア工学

実証的ソフトウェア工学 [4] とは，ソフトウェア開発現場での作業や実績に対する計測，定量化とその評価，そしてフィードバックによる改善という実証的手法を行う研究分野である．ソフトウェア開発の課題である生産性の向上や品質の確保に対する有用なアプローチとして注目されている．

2.5.1 妥当性についての議論

実証的ソフトウェア工学では，実在するソフトウェア開発データを用いたケーススタディを通じて提案手法の評価が行われることが多いが，このとき妥当性に関する議論が行われなければならない．妥当性については以下の分類 [16] が存在する．

内的妥当性
: 研究成果が研究の際に操作した要因から影響を受けている程度を指す

外的妥当性
: ある研究から得られた成果を，違った母集団，環境，条件へ一般化し得る程度を指す

構成概念妥当性
: 結果を得るために行った操作が適切である程度を指す

信頼性
: 他者が同様の手順で研究を行った場合，研究結果が再現可能となる程度を指す

実証的ソフトウェア工学における妥当性に関する研究として，文献 [17] の研究がある．この研究では，見積もり研究において研究成果の外的妥当性がどの程度意識されているかを調査するため，過去の研究論文を対象とした網羅的なレビューが行われている．調査の結果，対象となる 89 本の論文のうち，研究成果の外的妥当性についての議論を行っていない研究論文が 26 本存在しており，結論部分においてのみ言及している論文が 31 本存在することが示されている．このことから，見積もりの研究に携わる研究者は，研究成果の外的妥当性に関してより注意を払うべきであるということが主張されている．

2.5.2 追試 (replication study)

実証的ソフトウェア工学の分野では，既存研究で得られた知見に対する追試 (replication study) が重要であるとされている [5, 6]．研究に対する追試とは，ある研究に関して，実験の条件や環境を部分的に変更して実験を再現することである．追試を行うことによって，特定の条件において得られた知見が別の条件においても再現できるか，異なる条件や環境では別の知見が得られるか等の事柄を調査することができる．

追試によって異なる知見が得られた場合は，結果が異なった原因の考察や追加の調査に繋げることができる．同様の知見が得られた場合は既存研究の研究成果の外的妥当性がより高められたと判断することができる．また，追試を行い，様々な条件から得られた知見を統合することで，新規開発プロジェクトにおいてそれと類似した条件での知見を再利用することが可能となる．

3. 追試の準備

本章では追試を行うために必要な準備事項について述べる．

3.1. 論文の選択

ISBSG データが見積もり研究において一般的に用いられていることから，過去 5 年間での以下に示す主要国際会議，論文誌において見積もりに関する研究が行われている 83 本の論文のうち，ISBSG データが用いられている 22 本の論文を追試の対象とした．

- ACM-TOSEM (ACM Transactions On Software Engineering and Methodology)
- APSEC (ASIA-PACIFIC Software Engineering Conference)
- ESEM (Empirical Software Engineering and Measurement)
- ICSE (International Conference on Software Engineering)
- ICSM (International Conference on Software Maintenance)
- IEEE-TSE (IEEE Transactions on Software Engineering)
- IST (Information and Software Technology)
- JSS (Journal of Systems and Software)

追試を行うには，提案手法の詳細や実験手順，データの選別基準といった実験を再現するために必要となる情報が正確に記載されていなければならない．今回はそれらの情報が記載されている 4 本の文献 [7]，[8]，[9]，[10] を追試の対象として選択した．選択した論文をそれぞれ，生産性に基づくモデル作成に関する論文 [7]，カテゴリ変数の扱い方に関する論文 [8]，FP の簡易推定に関する論文 [9]，ニューラルネットワーク (以降，NN) を利用した見積もりに関する論文 [10] と名称付けて説明する．

3.2. 生産性に基づくモデル作成に関する論文の概要

回帰分析に基づく工数見積もりモデルに関する研究が行われており，対象プロジェクトの生産性も考慮に加えた工数見積もりモデルの作成方法を提案している [7]．ここでの生産性は以下の式 (7) によって定義される．

$$\text{生産性} = \frac{\text{FP}}{\text{工数}} \tag{7}$$

通常，工数見積もりモデルを作成する際にはデータセットに蓄積されたデータ全てを対象として回帰分析を行い，1 つのモデルを作成する．提案手法ではデータセットを生産性の高さによって複数に分類し，それぞれからモデルを作成する．そしてテストデータが持つ生産性の値の高さに基づき，複数のモデルのうち 1 つを選択して工数の見積もりを行う．現場で提案手法を実施する際，現行プロジェクトの生産性の推測はプロジェクトマネージャが行う．プロジェクトマネージャが生産性の推測を誤る確率を「推測誤り率」とする．

実験では，以下に示す 3 種類のモデルを作成し，見積もりの精度を比較する．Two レベルモデルを用いて対象プロジェクトの見積もりを行う場面を図 1 に示す．

No レベルモデル
: 生産性を考慮せずに回帰分析を行う従来の見積もり手法を用いるモデル

Two レベルモデル
: 生産性の値の高さによってデータセットを High(高), Low(低) の 2 段階に分類して提案手法を用いるモデル

Three レベルモデル
: 生産性の値の高さによってデータセットを High(高), Medium(中), Low(低) の 3 段階に分類して提案手法を用いるモデル

リリース 9 の ISBSG データ内の 593 データを対象として実験を行った．その結果，推測誤り率が 38 %以下

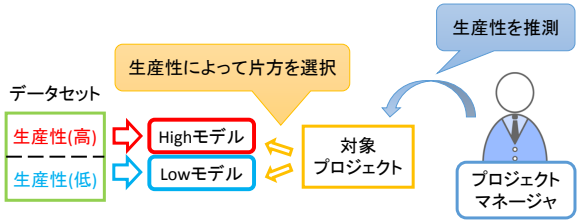


図 1. Two レベルモデルを用いた見積もり手法

の場合，つまり現場のプロジェクトマネージャが生産性の推測を誤る可能性が低いと判断できる場合には，従来の手法よりも提案手法の方が工数見積もりにおける見積もり精度が向上するという知見が得られた．

3.3. カテゴリ変数の扱い方に関する論文の概要

工数見積もりにおけるカテゴリ変数の扱い方に関する研究が行われている [8]．カテゴリ変数とは性別，職業など一般に数や量で測れない変数を指す．回帰モデルの説明変数としてカテゴリ変数を使用する際には，対応方法の異なる様々なモデルが用いられる．今回は以下の 4 種類のモデルを対象とする．これらの工数見積もりモデルを，カテゴリ変数を使用しないモデルと比較することで評価する．

ダミー変数化を用いたモデル

カテゴリ変数から複数のダミー変数を生成し，それらを説明変数として回帰モデルを作成する．プロジェクト p_i がカテゴリ変数 m_j においてカテゴリ値 c に属するかどうかを表すダミー変数 $d_{ij}(c)$ は，以下の式 (8) で定義される．

$$d_{ij}(c) = \begin{cases} 1 & (\text{カテゴリ } c \text{ に属する}) \\ 0 & (\text{カテゴリ } c \text{ に属さない}) \end{cases} \quad (8)$$

層別を用いたモデル

各カテゴリ変数の値の組み合わせにより，データセット内のデータをサブセットに分割する．そしてそれぞれのサブセットから回帰モデルを作成する．

交互作用モデル

ある説明変数の値によって，他の説明変数の効果が変化することを交互作用と言う．今回はダミー変数化を用いたモデルに，各ダミー変数と FP の積により作成した説明変数を加えたモデルを作成する．

階層線形モデル

グループごとにまとまりがあるデータを分析する際に用いられる．層別によって分割したサブセット間の関係性を考慮に加えて，サブセットごとに切片と傾きが変化するモデルを作成する．

リリース 9 の ISBSG データ内の 558 データを対象として実験を行った結果，カテゴリ変数を使用する 4 種類のモデルはいずれもカテゴリ変数を使用しない場合と比べて精度が約 10 %向上した．また，4 種類のモデル間での精度の差は 5 %未満となり，見積もりにおいて同程度の精度が得られた．

3.4. FP の簡易推定に関する論文の概要

FP の計測方法に関する研究が行われており，従来よりも簡易化された FP 推定モデルを提案している [9]．FP を計算する際に抽出される 5 種類の基本機能要素のうち，FP と最も相関の強い要素の規模のみを説明変数とした簡易 FP 推定モデルを作成する．相関関係の調査にはケンドールの順位相関係数とスピアマンの順位相関係数を用いる [18]．

リリース 11 の ISBSG データが持つ 600 以上のデータを対象として実験を行った結果，EI におけるケンドールの順位相関係数が 0.658，スピアマンの順位相関係数が 0.839 となり，FP と最も相関が強くなった．

3.5. NN を利用した見積もりに関する論文の概要

NN [19] を利用したソフトウェア開発期間の見積もりモデルの精度を調査している [10]．NN とは，人間の脳が問題を解く際の振る舞いを計算機上のシミュレーションによって表現したネットワークモデルである．今回は以下の 2 種類の NN である MLP (Multilayer Perceptron) と RBFNN (Radial Basis Function Neural Network) を使用する [20]．これらのモデルの精度を重回帰モデルを使用した場合と比較して調査する．

MLP

内部のニューロンが入力層，中間層，出力層に分かれており，ループせず単一方向にのみ信号が伝播するネットワーク

RBFNN

MLP の中間層で放射基底関数を用いて出力を計算するネットワーク

リリース 11 の ISBSG データ内の 49 データを対象として実験を行った結果，2 種類の NN モデルはいずれも重回帰モデルと比べて精度が 6%以上高くなった．このことから，NN を利用することによって開発期間の見積もり精度をより高めることができるといえる．

3.6. 使用する統計ツール

追試の中で回帰分析等の統計処理を行う際は，統計分析のフリーソフトである R [21] を使用する．

4. 追試の結果

本章では，対象となる 4 本の論文に対して IPA/SEC データを用いて追試を行った結果及び考察について説明する．

4.1. 生産性に基づくモデル作成に関する論文

4.1.1　データの選別基準

IPA/SEC データに蓄積されているデータのうち，表 1 に示す既存研究と同様の選別基準に従って抽出された 189 データを使用する．「本データの信頼性」とは，当該プロジェクトデータの合理性や整合性に関する信頼度を 4 段階 (A～D) で評価した値であり，最も信頼度が高い場合は A と評価される．

4.1.2　結果と考察

各モデルの説明変数には FP の対数，主開発言語グループ，開発プロジェクトの種別，開発対象プラットフォームのグループを用いる．追試における実験の結果を表 2 に示す．Two レベルモデルと Three レベルモデルの数値は，推測誤り率が 0 %の時点での数値である．さらに，推測誤り率を増加させた際の見積もり精度の推移を図 2，図 3 に示す．図 2 は横軸に推測誤り率が，縦軸に MBRE の数値が示されている．図 3 は横軸に推測誤り率が，縦軸に MdBRE の数値が示されている．

表 2 を見ると，推測誤り率が 0 %の時はいずれの評価指標でも No レベルモデルより Two, Three レベルモデル

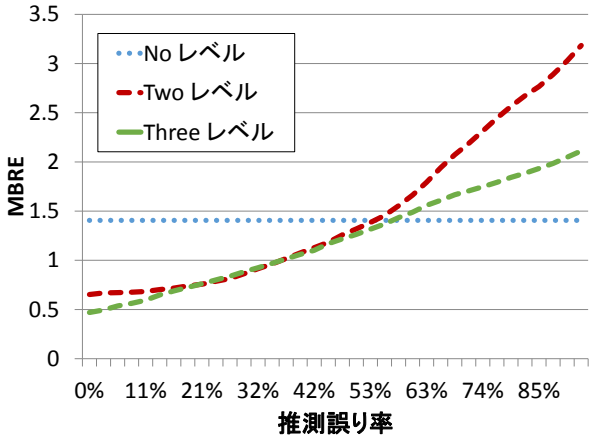


図 2. 追試における推測誤り率の増加による MBRE の推移

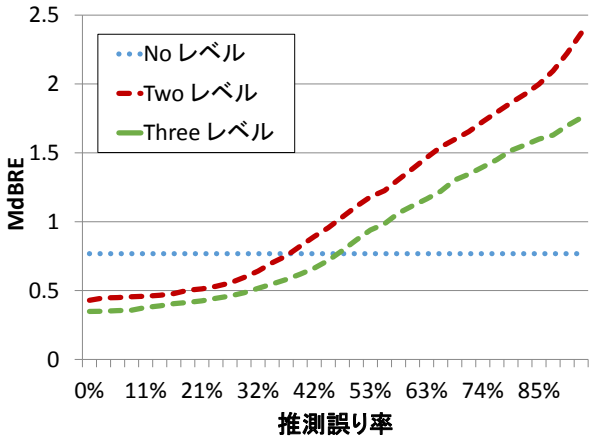


図 3. 追試における推測誤り率の増加による MdBRE の推移

の方が見積もりの精度が高い．図 2，図 3 を見ると，推測誤り率が増加するほど Two, Three レベルモデルの精度は低下している．しかし，推測誤り率が 37 %以下の時点では，MBRE と MdBRE のどちらにおいても Two, Three レベルモデルの方が No レベルモデルより精度が高い．

以上のことから，推測誤り率が低い状態，つまり現場のプロジェクトマネージャが生産性の推測を誤る可能性が低いと判断できる場合には，従来の手法よりも提案手

表 1. 生産性に基づくモデル作成に関する論文での IPA/SEC データの選別基準

FP 実績値の計測手法	IFPUG
本データの信頼性	A もしくは B
FP 実績値 (調整前)	欠損していない
主開発言語グループ	欠損していない
開発プロジェクトの種別	欠損していない
開発対象プラットフォームのグループ	欠損していない

表 2. 生産性に基づくモデル作成に関する論文の追試結果

レベル	MMRE	MdMRE	MMER	MdMER	MBRE	MdBRE
No	1.00	0.53	0.85	0.54	1.41	0.77
Two	0.50	0.33	0.46	0.35	0.65	0.43
Three	0.38	0.28	0.35	0.29	0.47	0.35

法の方が工数見積もりにおける見積もり精度が高くなると考えられる．これは既存研究から得られた知見と同様の知見である．

海外からデータを収集した ISBSG データと日本からデータを収集した IPA/SEC データで同様の知見が得られたことから，追試を行うことにより既存研究の研究成果の外的妥当性が高められたといえる．

4.2. カテゴリ変数の扱い方に関する論文

4.2.1 データの選別基準

IPA/SEC データに蓄積されているデータのうち，既存研究と同様に，まず表 1 に示す選別基準に従ってデータを抽出する．そして 3 つのカテゴリ変数 (主開発言語グループ，開発プロジェクトの種別，開発対象プラットフォームのグループ) の組み合わせからデータを分類し，10 データ未満のグループを省いた 88 データを使用する．

4.2.2 結果と考察

各モデルの説明変数には FP の対数と，カテゴリ変数である主開発言語グループ，開発プロジェクトの種別，開発対象プラットフォームのグループを用いる．追試における実験の結果を表 3 に示す．カテゴリ変数を使用しないモデルと，カテゴリ変数への対応方法が異なる 4 種類のモデルの見積もり精度が示されている．

表 3 から，4 種類のモデルはいずれもカテゴリ変数を使用しない場合と比べて精度が向上している．また，4 種類のモデルの精度に特徴的な差は無く，見積もりにおいて同程度の精度が得られる．

この結果を既存研究から得られた結果と比較すると，類似した傾向を示しており，追試によって既存研究と同様の知見が得られたと判断できる．このことから，生産性に基づくモデル作成に関する論文と同様に既存研究の研究成果の外的妥当性が高められたといえる．

4.3. FP の簡易推定に関する論文

4.3.1 データの選別基準

IPA/SEC データに蓄積されているデータのうち，表 4 に示す既存研究と同様の選別基準に従って抽出された 122 データを使用する．

4.3.2 結果と考察

追試における実験の結果を表 5 に示す．ケンドールの順位相関係数とスピアマンの順位相関係数を用いた際の，FP と各基本機能要素の相関係数の値が示されている．

既存研究で ISBSG データを用いた場合，FP と最も相関の強い要素は EI であるという結果が得られている．しかし表 5 より，IPA/SEC データを用いた場合における FP と最も相関の強い要素は，ケンドールの順位相関係数の

表 3. カテゴリ変数の扱い方に関する論文の追試結果

評価指標	カテゴリ変数無し	ダミー変数	層別	交互作用	階層線形
MBRE	111.2%	103.3%	107.7%	107.7%	103.4%
MMRE	85.5%	75.8%	78.8%	78.8%	75.5%

表 4. FP の簡易推定に関する論文での IPA/SEC データの選別基準

本データの信頼性	A もしくは B
EI	欠損していない
EO	欠損していない
ILF	欠損していない

場合は EI，スピアマンの順位相関係数の場合は EQ である．このことから，FP と相関の強い基本機能要素は必ずしも EI ではなく，データセットにより異なることがわかる．

よって，基本機能要素を用いて簡易 FP 推定モデルを作成する際は，使用するデータセットごとに適した要素を選択する必要があると考えられる．

4.4. NN を利用した見積もりに関する論文

4.4.1 データの選別基準

IPA/SEC データに蓄積されているデータのうち，表 6 に示す既存研究と同様の選別基準に従って抽出された 36 データを使用する．

4.4.2 結果と考察

各モデルの説明変数または NN の入力値には調整済みアプリケーション FP の対数と社内ピーク要員数の対数を用いる．追試における実験の結果を表 7 に示す．なお，リリース 11 の ISBSG データを用いて実験の再現を行った結果，こちらの準備した実験環境で NN モデルを用いると，一部のデータで本来の値と大きく外れた異常な予測値が発生しやすくなった．MdAR と Pred(25) は MAR よりも外れ値に対して頑健であることから，追試では MdAR と Pred(25) の値からモデルの精度を比較する．

表 7 を見ると，Pred(25) と MdAR のどちらにおいても重回帰モデルより精度が向上している NN モデルは RBFNN モデルのみである．これは既存研究と異なる結果である．

次に，追試では MLP モデルの精度が重回帰モデルより低下した原因を考える．既存研究では NN モデルの入出力値の関係性を調査している．その際，入力値である調整済みアプリケーション FP の対数と社内ピーク要員数の対数を説明変数とし，出力値である開発期間の対数を目的変数とする重回帰モデルを，データセットに蓄積された全てのデータに対して回帰分析を行い作成している．そのモデルが以下の式 (9) である．AFP (Adjusted Function Points) は調整済みアプリケーション FP を，MTS (Max Team Size) は社内ピーク要員数を，Duration は開発期間を指す．このモデルの決定係数は 0.560 となった．

$$\begin{aligned}\ln(\text{Duration}) = 0.15 + 0.438 \times \ln(\text{AFP}) \\ -0.187 \times \ln(\text{MTS})\end{aligned} \tag{9}$$

追試においても同様のモデルを作成した．そのモデルが以下の式 (10) であり，決定係数は 0.398 となった．

$$\begin{aligned}\ln(\text{Duration}) = -0.669 + 0.435 \times \ln(\text{AFP}) \\ -0.205 \times \ln(\text{MTS})\end{aligned} \tag{10}$$

既存研究で ISBSG データから得られたモデルの方が追試で IPA/SEC データから得られたモデルより決定係数が高いことから，入出力の相関関係は IPA/SEC データより ISBSG データの方が強いことがわかる．また，NN モデルは入出力値の相関関係が強いほど予測精度が高くなるとされている [22]．さらに，RBFNN モデルは MLP モデルより安定した学習が可能であるとされている [23]

表 5. FP の簡易推定に関する論文の追試結果

要素	ケンドール	スピアマン
EI	0.663	0.836
EO	0.607	0.791
EQ	0.656	0.850
ILF	0.655	0.837
EIF	0.384	0.525

表 6. NN を利用した見積もりに関する論文での IPA/SEC データの選別基準

本データの信頼性	A もしくは B
実績月数 (開発期間)	2ヶ月 以上
社内ピーク要員数	2 人 以上
開発対象プラットフォームのグループ	Windows 系
主開発言語	第 3 世代言語
開発プロジェクトの種別	新規開発

．以上のことから，既存研究よりも入出力値の相関関係が弱くなった影響を MLP モデルのみが受け，MLP モデルの見積もり精度が低下したと考えられる．

5. おわりに

本稿では，ISBSG データが評価に用いられている見積もりの既存研究に対する追試を実施した．過去 5 年間の論文から 4 本の論文を対象とし，IPA/SEC データを用いて追試を行った．その結果，2 本の論文では既存研究と類似した知見が得られたことから，既存研究の研究成果の外的妥当性が高められたという結論が得られた．残りの 2 本の論文では既存研究と異なる知見が得られたことから，その原因に対する考察を行った．

今後の課題としては，まず見積もり分野における他の既存研究に対する追試が考えられる．実証的ソフトウェア工学では妥当性への脅威に関する議論が重要となるが，その必要性に対する認識は未だに不十分である．よって，今回のような妥当性に関する議論を継続的に行うことで，その必要性に対する認識をより広めていくべきである．また，今回の追試から得られた知見に対する追加調査も今後の課題として考えられる．例えば，FP の簡易推定に関する論文について，各業種と導かれた要素との関連性に対する調査が挙げられる．さらに，外的妥当性をより高めるために，今回用いたデータセットとは異なる種類のデータセットで追試を行うことも必要であると考えられる．

6. 謝辞

本研究を行うにあたり，データを提供して頂くとともに多大なご助言を頂きました 独立行政法人情報処理推進機構技術本部ソフトウェア高信頼化センターの 関係各位 に深く感謝申し上げます．

本研究は一部，日本学術振興会科学研究費補助金基盤研究（S）（課題番号：25220003）の支援を受けている．

参考文献

[1] J. Radatz, A. Geraci, and F. Katki. IEEE standard glossary of software engineering terminology. *IEEE Std*, 610.12-1990.

[2] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches - A survey. *Annals of software engineering*, 10, 1-4, 177–205, 2000.

[3] International Software Benchmarking Standards Group (ISBSG). http://www.isbsg.org.

表 7. NN を利用した見積もりに関する論文の追試結果

モデル	ニューロン数	MAR	MdAR	Pred(25)
重回帰	-	0.33	0.27	0.44
MLP	5	0.44	0.30	0.50
RBFNN	14	0.37	0.20	0.53

[4] B. Kitchenham, et al. Evaluating guidelines for reporting empirical software engineering studies. *ESE*, 13, 1, 97–121, 2008.

[5] F. QB Da Silva, et al. Replication of empirical studies in software engineering research: a systematic mapping study. *ESE*, 19, 3, 501–557, 2014.

[6] F. J Shull, J. C Carver, S. Vegas, and N. Juristo. The role of replications in empirical software engineering. *ESE*, 13, 2, 211–218, 2008.

[7] M. Tsunoda, A. Monden, J. Keung, and K. Matsumoto. Incorporating expert judgment into regression models of software effort estimation. In *APSEC*, 1, 374–379. IEEE, 2012.

[8] M. Tsunoda, S. Amasaki, and A. Monden. Handling categorical variables in effort estimation. In *ESEM*, 99-102. ACM, 2012.

[9] L. Lavazza, S. Morasca, and G. Robiolo. Towards a simplified definition of Function Points. *IST*, 55, 10, 1796–1809, 2013.

[10] C. López-Martín and A. Abran. Neural networks for predicting the duration of new software projects. *JSS*, 101, 127–135, 2015.

[11] Barry W Boehm. *Software engineering economics*. 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[12] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *TSE*, 29, 11, 985–995, 2003.

[13] IFPUG: Function Point Counting Practices Manual, Release 4.3. *IFPUG*, 2010.

[14] 柏本, 楠本, 井上, 鈴木, 湯浦, 津田. イベントトレース図に基づく要求仕様書からのファンクションポイント計測手法. 2000.

[15] IPA 独立行政法人 情報処理推進機構. https://www.ipa.go.jp.

[16] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.

[17] 江川翔太. 見積もり研究における外的妥当性の調査を目的とした系統的レビューと追試. 修士学位論文, 大阪大学, 2016.

[18] C. Croux and C. Dehon. Influence functions of the Spearman and Kendall correlation measures. *Statistical methods & applications*, 19, 4, 497–515, 2010.

[19] Richard P Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4, 2, 4–22, 1987.

[20] J. Park, R. G Harley, and G. Kumar Venayagamoorthy. Adaptive-critic-based optimal neurocontrol for synchronous generators in a power system using MLP/RBF neural networks. *TIA*, 39, 5, 1529–1540, 2003.

[21] R: The R Project for Statistical Computing. https://www.r-project.org.

[22] 一言, 桜庭, 小野. ニューラルネットワークを用いた洪水予測システムの開発. こうえいフォーラム: 日本工営技術情報, 20, 67–72, 2012.

[23] Masayuki Murakami. *Practicality of modeling systems using the IDS method: Performance investigation and hardware implementation*. PhD thesis, The University of Electro-Communications, 2008.

週報のテキストマイニングによるリスク対応キーワード抽出

野々村　琢人
(株)東芝　インダストリアルICTソリューション社
takuto.nonomura@toshiba.co.jp

安村　明子
(株)東芝　インダストリアルICTソリューション社
akiko.yasumura@toshiba.co.jp

弓倉　陽介
(株)東芝　インフラシステムソリューション社
yosuke.yumikura@toshiba.co.jp

要旨

ソフトウェアの研究開発管理において，週報などの報告書による情報は重要であるが，以下の課題がある．部門内のチームリーダによる集約された週報では情報が欠落してしまう．逆に部門メンバー全員の週報は膨大で冗長な表現も多く，短時間での確認が困難である．

今回，メンバー個人の週報および部門全員の週報に対してテキストマイニングを行い，リスク対応に必要なキーワードを抽出できることを確認した．

1. はじめに

ソフトウェアの研究開発において，担当者は週報（間隔によって，日報，半月報，月報などもある）を管理者に提出する．管理者は報告会などで直接担当者から情報を得る以外に，提出された週報から進捗やリスクを確認する．

部門メンバーが多い場合には，個人の週報ではなくチーム毎にチームリーダが纏めた週報を管理者は読む．しかし，チーム内の複数案件の情報を報告するため，チームリーダが適切に情報を集約しないと重要な情報が欠落する可能性がある．さらに週報にはフォーム（形式）を決めにくい，あるいは決めても守られないという傾向があり，個人ごとの週報の場合，そこから課題やリスクに繋がる情報を確実に読み取るには，時間がかかる．

そこで，主なトピックスや概要はチームリーダ週報で把握し，気になる／気にすべきトピックスを必要に応じて担当者個人から直接ヒアリングしたいという管理者の希望がある．

近年，アンケートなどの自由記述の回答データの分析にはテキストマイニングが利用されている．規則性のない非構造化データである「週報」を分析するのに，このテキストマイニング技術が有効であると考え，試行実験を行ったので，それを報告する．

なお本稿では「リスク」とは「それが発生（顕在化）すると影響を与える不確実な事象・状態」[1]とする．

2. 本試行実験の方法

リスクを発見し，その問題を事前に解決するため，次のようなステップを考える（図1）．Step1　週報を入力としたテキストマイニングによりリスク発見に繋がるキーワードを抽出，Step2　抽出したキーワードを元に担当者へのヒアリング，Step3　ヒアリングに基づく課題対応．

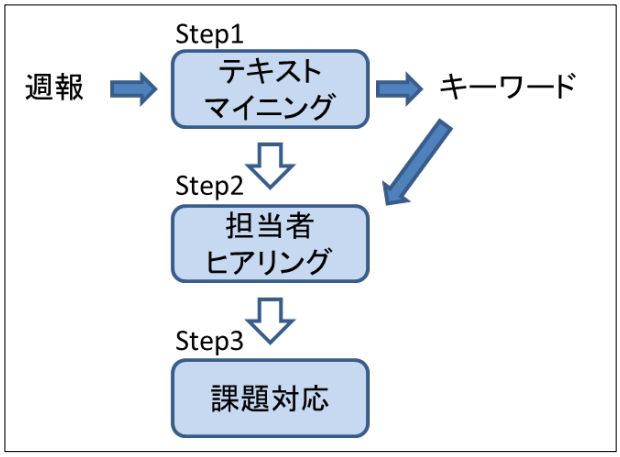


図1：リスク対応ステップ内のテキストマイニング

今回この Step2 の担当者ヒアリングに利用可能なキーワードを Step1 で効果的に抽出できることを確認するため，次の2つの実験を行った．

実験1：個人週報からキーワードを抽出

週報などの報告資料ではリスクを早めに検知し回避することが重要である．しかし悪い報告は遅れがちであり，特にトラブルかどうかの判断が難しい情報を担当者やチ

ームリーダがトラブルでないという情報としてしまうこともある．そこでチームリーダが集約したチーム週報と，担当者の個人週報をテキストマイニングで分析する．それにより管理者からみてリスクかもしれない状況を発見するためのキーワード抽出にどの程度効果があるかを確認する．

実験2：部門全員の週報からキーワードを抽出

部門内にメンバーが多い場合，管理者が全員の週報を確認する時間を確保することは容易でなく，部門を分割したチームの週報で把握することも多い．チームリーダが自チームの週報を集約することで消えてしまった「リスク発見に繋がるキーワード」を全員の週報から発見すること，及び全員の週報を俯瞰的にみて，チーム間での調整が必要なことなどに関する新たな表現に気づくことが可能かを確認する．

3. 個人週報からキーワードを抽出（実験1）

今回の実験では計量テキスト分析やテキストマイニング専用のフリーソフトウェア「KH Coder」[2][3]を利用した．これは「茶筅」[4]などの形態素解析システムにより語を抽出し，語の繋がりの強さなどを統計的に解析するソフトウェアである．アンケートの自由回答項目やインタビューデータなどのテキスト分析に利用されている．

3.1. 実験内容

・利用データ：次2つのデータを利用

(1) 担当者（1 人）の個人週報
［6 ヶ月間 22 回分 543 行］

(2) チームリーダ(1 人)のチーム週報
［6 ヶ月間 22 回 2007 行］

・分析手法：共起ネットワーク分析（Graph 理論に基づき，関係の深さを頂点と辺により表現するもの．共起パターンを使うことで語の繋がりを判断する）及び，文書検索機能

・実験手順：

(1) 個人週報を入力とし，共起ネットワーク分析を実施
(2) チーム週報を入力とし，共起ネットワーク分析を実施
(3) 個人週報の共起ネットワーク図からヒアリングに利用したいキーワードを管理者が視覚的に選定
(4) 選定したキーワードが，チーム週報の共起ネットワーク図に抽出されているかを確認

・KH coder の設定：

最低出現回数の設定＝ (1)チーム週報の時 3 回及び 2 回 (2)個人週報の時 3 回
（最低出現回数の設定値が高すぎると汎用的な語しか抽出できず，低すぎると抽出される語が多すぎ特徴をつかめない）

3.2. 実験結果

共起ネットワーク分析により出現頻度が高い語として抽出された語数は表 1 のようになった．また，個人週報の共起ネットワーク分析結果は 68 個がグループ化され，図 2 のように表示される．この図は出現頻度が高い語と，それらの語間の繋がりの強さを示している．

表1：共起ネットワーク分析での出現語数

	最低出現回数設定	グループ数	語数
チーム週報	3	２１	１１６
	2	３９	３７１
個人週報	3	１５	６８

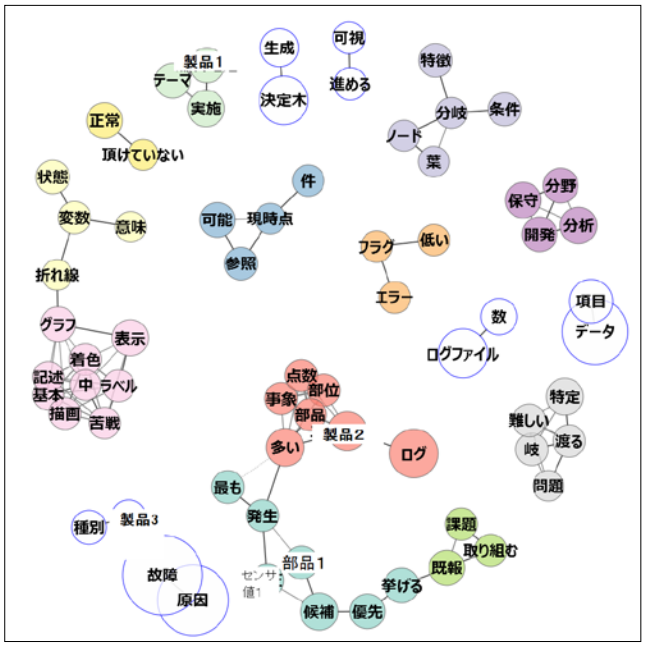


図 2：個人週報の共起ネットワーク分析

この共起ネットワーク図から個人週報のヒアリングに利用可能なキーワード（センテンス）として視覚的判断したものは以下の 3 件である．

(1)「（まだ）正常な（データを）頂けていない」
(2)「ラベル描画に苦戦中」
(3)「問題が多岐にわたり特定が難しい」

また，「故障」「原因」という語も頻出しているが，今回．

これらの語を含むセンテンスはヒアリングのキーワードとして選ばなかった．その理由は，この個人週報のテーマの一つが「故障原因の候補を早く見つける手法」であり，テーマ名として，この2語が頻出しているためである．

次にチーム週報を同様に分析した結果，最低出現回数設定＝3 回のとき，個人週報で抽出したキーワードのうち(1)のみ存在し，(2)と(3)は表示されなかった．(表 2)

さらにチーム週報の最低出現回数設定＝2 回の場合，キーワードは出現したが，共起ネットワーク図に表れる語数が371となり，図としても円が重なり判別が非常に困難であった．これはチーム週報においては汎用的な技術用語が多くなり，リスクに繋がるキーワード出現数が相対的に下がるためと考えられる．

表 2 個人週報とチーム週報でのキーワード出現

管理者指摘のキーワード	個人週報	チーム週報 [設定 3 回]	チーム週報 [設定 2 回]
正常－頂けていない	○有り	○有り	○有り
ラベル－描画－苦戦中	○有り	×無し	○有り
問題－多岐－特定－難しい	○有り	×無し	○有り
判別可能な図か？	YES	YES	NO

このように，担当者の個人週報に対する共起ネットワーク分析は特徴的な語を抽出することができ，チーム週報の共起ネットワーク分析よりも効果的であることを確認できた．

3.3. 追加実験（テキスト目視との比較）

共起ネットワーク分析を利用せず，週報全体を全部読んだ場合との比較はどうなるかを，追加実験として行った．

対象データは(1)個人週報の共起ネットワーク図 6 枚，(2)チーム週報（リーダによる集約版），(3)個人週報 6 人分である．これらの所要時間とキーワード出現の有無を確認した（表 3）．共起ネットワーク図は事前準備に時間を取られるが，作成後の俯瞰は短時間ですむ（3.4 の文書検索による詳細確認を除く）．個人週報全員分の読み込みは時間がかかるだけでなく，キーワードが記載されていても読み落とすことも発生した．チーム週報全体では集約により，共起ネットワーク図で見つけたキーワードは含まれていなかった．その意味でチーム週報全体を閲覧することよりも，共起ネットワーク分析の利用は効果的であった．

表 3 追加実験（テキスト目視確認との比較）

対象のデータ	準備時間	確認時間	所定キーワードは出現したか
個人週報の共起ネットワーク図×6 枚（6 人分）	35 分	2 分	○出現
チーム週報全体テキスト（2007 行）	0 分	40 分	×非出現
個人週報 6 人分テキスト（3696 行）	0 分	60 分	○出現

3.4. 適用上の留意点

この試行実験結果を適用する場合の留意点の一つは，そのキーワードに関するヒアリングを至急行うべきかどうかの判断を行う前に，次のような 5W1H の情報を把握する必要があることである．

(a) どの程度の頻度（回数）で出現しているのか
(b) いつの週報に出現しているのか（また，最後に出現した週報はいつか）
(c) どの文脈で（どのセンテンスの中で，どの意味で）出現しているのか

これにより例えば「まだデータを頂けていない」という状態について，それがヒアリングに値するかどうかを判断する．

この確認には KH coder の機能の一つである「文書検索機能」を利用した．抽出されたキーワードを選ぶと，それを含むコンテンツ（該当するセンテンスや行）を表示する機能である．それにより今回のケースでは「データを頂けていない」という状態は 4 週間続き，それが現在は収束していることが分かった．

4. 部門全員の週報の分析（実験 2）

4.1. 実験内容

・実験データ:
(1) 部門全員(4 チーム計 34 人)の個人週報（マージ版）
［1 回分，1230 行］
(2) 各チームのチーム週報（4 チーム分）
［1 回分，平均 52 行／チーム］
・分析手法：対応分析，特徴度抽出（Jaccard 類似性）
・実験手順:

(1) 個人週報(マージ版)を入力とし対応分析を実施
(2) 分析出力の分布図から，ヒアリングに利用したいキーワードを管理者が視覚的に選定
(3) 各チームのチーム週報に，選定したキーワードが含まれるかを検索で確認
(4) チームの特徴語を Jaccard 類似性測度で抽出し，チーム週報に含まれるかを確認
(5) 分布図と上位出現リスト表(30 語)の比較

・KH coder の設定： 最低出現回数の設定＝18 回

4.2. 実験結果

部門全体の週報から 98 個のキーワードが取り出され，4 チームの特徴が分布図で視覚化された(図 3)

この図の中心にある語は特徴の少ない語であり，横軸の左右または縦軸の上下に寄った語ほど，特定のチームを特徴付ける語となる．

例えば「実装，機能，モジュール」などの語はどのチームの週報にも表れ，他の語との関連性も弱いことが分かる．また，左上のチーム 1 を特徴づける語は「環境，ビルド」などであることが分かる．このように特定のチームに出現する語の例は表 4 のようになる．

表 4 抽出された語が含まれる分布図のチームエリア

語	チーム 1	チーム 2	チーム 3	チーム 4
1.実装	○	○	○	○
2.機能	○	○	○	○
3.モジュール	○	○	○	○
4.環境	◎			
5.ボード	◎			
6.設備		◎		
7.アダプタ		◎		
8.特許			◎	
9.閾値			◎	
10.ヒアリング				◎
11.バックアップ				◎

(○:全チームで出現，◎:特定チームで出現)

この分布図を俯瞰し，ヒアリングに利用したいキーワードとしては以下を選定した． チーム 1:「ライセンス」， チーム 2:「用語 3」， チーム 3:「特許」， チーム 4:「ヒアリング」．

キーワードとして選定したのは次のような理由である，チーム 1 の「ライセンス」は，ライセンス契約で気になることがあるのか？の確認を行いたい．チーム 2 の技術用語「用語 3」はこれまで報告のなかった用語でその発生元を含め確認したい．チーム 3 の「特許」は特許出願を準備中か他社特許の調査を行っていると推測できるが，特許関連で何を行っているのか？問題はないか？などを確実にフォローしたい．またチーム 4 では「ヒアリング」というキーワードがある．もしヒアリングが既に終わっている予定であれば，担当の今週の週報にあがってくるのは，まだ何かが続いている可能性があり要注意である．あるいは単に過去のヒアリング結果に対する分析なのか，などを確認する必要がある．

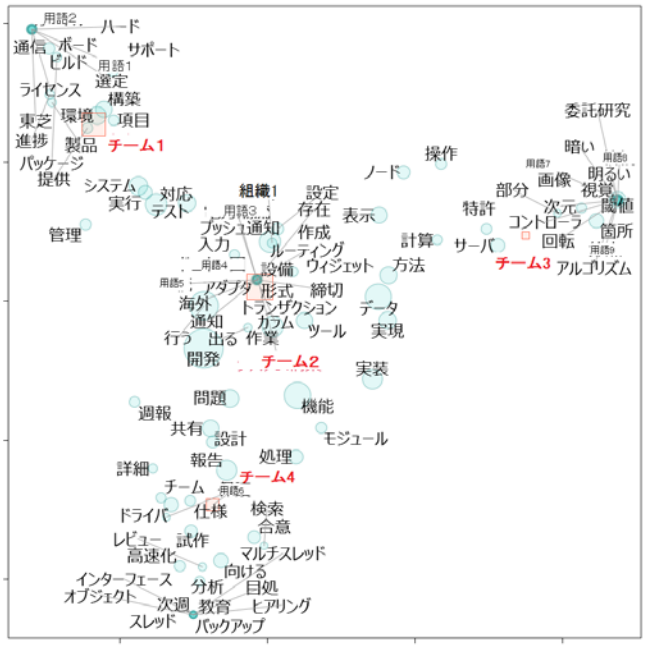


図 3 部門全体の週報(個人週報全員分)の対応分析

図 3 を確認して選定した 4 つのキーワードが，各チームのチーム週報に記載されているかどうかも確認した，その結果は次の通りである．

チーム 1 週報:「ライセンス」・・×(なし)
チーム 2 週報:「用語 3」・・・・・○(あり)
チーム 3 週報:「特許」・・・・・・×(なし)
チーム 4 週報:「ヒアリング」・・○(あり)

また，各チームを特徴づける語として，「Jaccard の類似性測度」(以下測度)を計算し各チーム 6 語をリストアップした．これはデータ全体に比べ，それぞれのチーム週報において特に高い確率で出現している度合いを示す数値である．更にその特徴的な語が各チームの週報内に出現するかを確認し，ヒット率を計算した(表 5)．

（ヒット率＝ヒットした語の測度合計／6 語の測度合計）

表 5 各チームの特徴語

	特徴語	測度	出現有無	ヒット率
チーム1	環境	0.072	○	
	構築	0.059	×	
	システム	0.047	○	
	ビルド	0.042	×	
	ボード	0.028	○	
	パッケージ	0.028	×	
	チーム1ヒット率			53.3%
チーム2	ツール	0.029	×	
	通知	0.026	○	
	設備	0.019	×	
	ウイジェット	0.019	×	
	アダプタ	0.017	×	
	用語4	0.017	×	
	チーム2ヒット率			20.5%
チーム3	視覚	0.088	×	
	閾値	0.044	○	
	画像	0.027	×	
	明るい	0.035	×	
	次元	0.031	×	
	特許	0.023	×	
	チーム3ヒット率			17.7%
チーム4	報告	0.059	×	
	ドライバ	0.035	×	
	試作	0.03	○	
	高速化	0.028	○	
	仕様	0.02	○	
	バックアップ	0.018	○	
	チーム4ヒット率			50.5%

（出現有無 ○:有り,×:無し）

このように，この部門のチーム週報はトピックスの一部のみを重点的に記載する傾向があるためか，特徴語は17.7%~53.3%しか含まれていなかった．

4.3. その他の確認結果

上位出現リストとの比較は，個人週報（全員分）の上位 30 語の出現リストの確認により行った（表 6）．リストで 84 回出現し 12 位に入っている「組織 1（固有名詞）」は部門としては関わりが深い組織となるが，分布図（図 2）で確認するとチーム 2 のみが関わっている組織であることが分かる．このように単純な出現リストより，対応分析の分布図の方が状況をより適切に示していると言う意味で，有効であると考えられる．

チーム間での調整などに関する新たな事実に気づくことが可能か？に関しては，同じ問題に遭遇しているケースや似たものを無駄に二重開発しているケースを想定した．

しかし今回，部門間での調整などが必要と思われる情報は特に抽出できなかった．これは当該部門の各チームの業務や特徴が大きく異なり，似た課題に遭遇するなどの状況が表れにくいことも影響していると考えられる．

表 6 実験 2 の抽出語と出現回数 上位 30 語

抽出語	出現回数	抽出語	出現回数
開発	207	対応	69
画像	153	確認	65
作業	144	組織 2	65
アルゴリズム	135	技術 1	64
部品	124	技術 2	63
ファイル	110	担当	61
データ	105	処理	60
作成	98	必要	58
ログ	95	実施	57
検出	93	最適	56
案件	88	検討	56
故障原因	84	適用	50
行う	84	報告	48
組織名1	84	予定	48
機能	73	可能	47

5. 課題と工夫点

5.1. 一般的なテキストマイニングでの課題と対応

本手法での課題とは別に，一般的なテキストマイニングでの前処理なども実際の運用では必要である．

（1）データクレンジング

テキストデータに対して以下を実施しておくことが分析精度に大きな影響を与えるので不可欠である．

a)誤記，誤字の修正

b)機種依存文字の削除，変更

c)半角，全角の統一

d)表記の揺れの修正

今回の試行では分析対象のテキストデータが週報のため，a)誤記・誤字 や b)機種依存文字 は殆ど無く，対応がほぼ不要であった．しかし c)半角・全角 および d)

表記の揺れは多く出現しており表現の統一を行った.

(2) 強制抽出(専門用語の指定)

自然言語処理の形態素解析では, 専門用語が分割されて認識されることがある. 例えば「ログファイル」が「ログ」と「ファイル」, 「ハードディスク」が「ハード」と「ディスク」などである. 用語を正しく抽出するために, 分割されてしまう専門用語を事前にピックアップし, KH coder の「強制抽出」設定を行った.

これにより本来意識したい用語を適切に抽出し, 分析の精度をあげることができる.

指定用語としては「プラットフォーム」「通信インターフェース」「ソースコード」などのソフトウェア関連の用語の他にも, プロジェクト内で使われる製品ドメイン固有の技術用語も指定した.

5.2. 本手法での課題と対策

先の 5.1(2)の強制抽出のための専門用語の指定は, 一般に抽出結果を見て専門用語の分割に気づき, 用語を指定する手順の繰り返しとなる. そのためソフトウェアの研究開発のように専門用語が多い週報では, 指定すべき用語を試行錯誤で決めるオーバヘッドが非常に大きい.

そこでソフトウェアの研究開発に関連する用語を Web 上の IT 用語辞典(e-Words[5]など)からピックアップして, 強制抽出したい用語の指定に利用した.

これらの IT 用語辞典では, 製品ドメイン固有の技術用語はカバーされないが, 一般的なソフトウェア関係の技術用語はほぼカバーされている. 語数は, プログラミング用語が 1068 件, ソフトウェア用語が 912 件, セキュリティ用語が 649 件などである. (e-Words の場合)

また, 分野をより絞ったカテゴリ, 例えば「ソフトウェアテスト」114 件などもあり, 対象とするチームの特性を考慮してカテゴリを選んだ.

これにより, 当初は強制抽出する専門用語の指定に 50 分程度かかっていたものが, 10 分程度に短縮することができた.

6. 評価と考察

6.1. 個人週報の分析(実験 1)

個人週報の共起ネットワーク分析では, データとしてある程度のボリューム(何週間分か)が必要であるが. 次のような効果を期待できる.

1.毎週続けて記載した(訴えた)情報はヒアリング用キーワードとして抽出が可能, 2.リスク分析には特に文意が重要であるが, KH coder 内のリンク機能で個別週報に戻らずに直ぐに内容を確認できる.

実験 1 の結果に関するアンケート(対象数 36 人)では以下のような評価を得た.

(1)使えそう／使ってみたい	14%
(2)(改善すれば)ある程度使えそう	59%
(3)あまり使えない	19%
(4)まったく使えない	08%

6.2. 部門全員の週報の分析(実験 2)

チームごとの特徴的キーワード抽出では, 部門の人数が多く, 生の週報を全部確認できない場合に利用すると次のような効果を期待できる. 1.確認やフォローの切り口を容易に見つけられる, 2.チーム固有の活動状況や課題を見つけられる(事例では特許, ヒアリング)

実験 2 の結果に関するアンケート(対象数 36 人)では以下のような評価を得た.

(1) 使えそう／使ってみたい	11%
(2)(改善すれば)ある程度使えそう	50%
(3)あまり使えない	28%
(4)まったく使えない	11%

6.3. 評価アンケートコメント

改善を前提に使えそう／ある程度使えそうという回答(上記の(1)と(2))が合計 61%～73%と, 一定の評価は得られた.

しかし「まったく使えない」という強いネガティブ評価も 8%～11%あり, 活用にはまだまだ改善が必要と思われる.

なお代表的なコメントは, 以下のようなものである.

[ポジティブコメント]

(1)週報によっては, 明るいニュースを優先し, 現在も残っている過去のリスクに触れないことがある. そのようなケースでもプロジェクトの不安から自身の週報に記載している個人週報がある. そこから見つけることを期待したい. (2)部門が 20 人, 30 人となった時には時間的制約から, チームリーダの週報のみを見ていたが, 全員の週報確認であってもこの分布図のボリュームなら使ってみたい.

[ネガティブコメント]

(1)クレンジングなどのオーバヘッド削減の見込みがないと, 使いにくい.

全体として. 特に大量の週報に対する俯瞰的分析が

期待をされている．しかしクレンジング作業のオーバヘッドがまだ大きく準備に時間を取られるため，その改善が利用の前提となる．

6.4. 考察

リスクに繋がるキーワード抽出以外の活用として，チーム間での情報交換や部門長と担当者のコミュニケーションにも効果があると思われる．（他チームの膨大な週報を読むことはあまり行われないため）

また，ソフトウェアの研究開発におけるリスク管理では進捗管理がよく利用されるが，進捗遅れが出る前にその原因となるリスクを見つけることが本来望ましい．本手法では「遅れた」と明記していない段階で気になるキーワードを見つけてヒアリングに繋げることが可能である．これがソフトウェアの研究開発において，週報のテキストマイニングを利用するメリットの一つである．

7. 課題と改善方針

7.1. 課題

今回の実験により，以下のような課題もリストアップされた．

1) 判断に関して
   ・肯定か否定か（成功か失敗）の判断には，動詞を含む詳細な分析が必要
   ・実験1では過去 6 ヶ月分の週報を用いたが，この分析でリスクに気づいた時には既に手遅れとなっている可能性がある．
2) 準備や整備に関して
   ・同じ用語で表現が異なる（例：インターフェース，インタフェース，Interfaceなど）と抽出に誤差が出るので，週報作成時に表現を統一することが望ましい
   ・専門用語が形態素解析で分割されることがないように，強制抽出設定を行う（5.1(2)）など，分析前のデータ整備の工数削減が必要
3) 運用に関して
   ・リスクを隠したいメンバーは該当するキーワードをNGワードとして利用しなくなる可能性もある

7.2. 改善方針

1) キーワード選定のためのデータセット準備

今回，分布図や対応図からキーワードを抽出する方法は，管理者が経験から気になるキーワードを選定する方法であった．しかし事前に「ソフトウェア研究開発の管理において注意すべきワード」をデータセットで用意しておけば，それを利用した抽出が可能になる．特にソフトウェアの研究開発管理という視点での抽出が実現する．

これはまた，管理者の経験に依存せず（異なる領域部署の管理者として門外漢が新たに着任した場合など）リスクに繋がるキーワードを容易にピックアップすることが可能になると期待される．

2) 強制抽出用の専門用語のデータセット準備

今回，強制抽出のための専門用語の指定では，IT 用語辞典を利用する工夫を行った．かなりの工数削減を実現できたが，上記同様にデータセットとして準備することで，精度を更に向上できる．すなわち研究開発対象のドメインの技術用語や製品用語を知識継承的にストックしておき利用する．

3) 語の出現回数の最適な設定

KH coder では語の抽出に関して「出現数」と「文書数」という設定がある．「出現数」は語がデータ全体を通して出現した回数を示し，「文書数」は語がいくつの文書中に出現したかを示す．出現数が極端に少ない語はキーワードとも言いがたく，また統計的分析に利用しにくい．そこで最小出現数を設定し，それ以下の語がノイズとして出現しないようにする．本実験 1 では 2 回 or3 回，実験 2 では 18 回の指定である．また，同じ出現数でも，文書数が多い語はどのコンテキスト（文脈）でも現れる語（例えば「ソフトウェア」）と見なせる．これに対しては，満遍なく出現する「キーワードとは言えない一般的な語」を除外するために，最大文書数を設定し出現数を制限する．これらの設定値は試行錯誤で行うことが多くオーバヘッドが大きくなる．

そこで，語の出現回数の最適な設定値を，部門の人数や週報のボリューム（総行数）から，デフォルト値として決めたい．つまり，部門 X（人数 n 人）で，その週報の記載の仕方（トレンド）から，最大・最小出現数の最適値を割り出しておく．

4) 週報表現の統一方法の工夫

週報の記載に際して，そのフォームや，専門用語（テクニカルターム）の書き方を統一することで，より精度の向上とオーバヘッド時間の短縮が期待できる．

8. 終わりに

8.1. まとめ

テキストマイニングはアンケート結果やコールセンターのクレーム内容などの分析に利用が進んでいる．またソ

フトウェアの研究・開発への適用としては LDA（Latent Dirichlet Allocation：潜在的ディレクトリ配分法）を利用したバグローカライゼーション分析などがある[6].

今回ソフトウェアの研究開発の報告書の一つである週報に対して，KH coder を使うテキストマイニングを用いて状況把握，リスク発見に繋がるキーワード抽出を行うことを試みた．頻出リストよりも見やすくセンテンスの特徴を表現する共起ネットワーク図や対応分析図がキーワード抽出に効果的であることを確認した．またアンケートでも条件付きながら 61%～73%の評価を得られた．

部門全員の週報分析（実験 2）では，6.評価と考察で示した効果の他に，今回は検証にいたらなかったが次の三点も期待される．

1) チーム間での不整合や対立の発見
2) 共通に出現する表現から，複数のチームでの二重開発やツールの二重購入などのムダの発見
3) 複数チームでの共通の課題（品質，コスト，納期など）の早期発見と横断的な解決

勿論，正確な情報分析を行うには，直接担当者に口頭やメールで確認するプロセスも必要である．

その問い合わせや確認のための「切り口」「キーワード」をテキストマイニングからスピーディーに得られれば，より担当者への確認が適切なものになる．

今後は，ソフトウェアの研究開発固有のデータセットの収集や，用語表現の統一などのオーバヘッド削減に関して引き続き検討し，ソフトウェア研究開発における他のドキュメントへの適用なども検証していきたい．

参考文献

[1] PMBOK®ガイド第 5 版
[2] 樋口耕一, KH Coder <http://khc.sourceforge.net/>
[3] 樋口耕一，社会調査のための計量テキスト分析，ナカニシヤ出版
[4] NAIST，形態素解析システム茶筅 http://chasen.naist.jp/hiki/ChaSen/
[5] IT 用語辞典 e-Words http://e-words.jp/
[6] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, “Bug localization using Latent Dirichlet Allocation,”Information and Software Technology vol. 52, no. 9, pp. 972–990, 2010.

開発の"待ち", "遅れ", "欠陥"を防ぐプロセス改善
PM 改善ナビを利用した"待ち", "遅れ", "欠陥"の改善活動

比嘉 定彦
株式会社アドバンテスト
sadahiko.higa@advantest.com

要旨

開発工程の中で発生する「待ち」や「遅れ」の状態をムダであるとし, (「欠陥」と同様に)改善する手法として, トヨタ開発方式[1]というのがある. 我々は, 必要な部分から順次カスタマイズし, 開発現場へ導入した. その結果, トヨタ開発方式は開発の「待ち」, 「遅れ」, 「欠陥」を未然に防ぐ有効な解決法であることを確認した[2][3][4].

開発の「待ち」, 「遅れ」, 「欠陥」の改善は, プロジェクト管理が扱う範囲全体から見ると一部の領域にすぎない. しかし, プロジェクト管理の実施効果は製品開発のQCD実績で把握される点を考慮すると, 開発の「待ち」, 「遅れ」, 「欠陥」を改善する活動領域は重要である.

我々は数年に渡るトヨタ開発方式の利用実績を基に, 開発の「待ち」, 「遅れ」, 「欠陥」を改善するための要点を体系化して「PM(ProjectManagement)改善ナビ(以降PM 改善ナビと記述する)」へ組込み, 開発現場で適用を試みた.

本報告では, 開発の「待ち」, 「遅れ」, 「欠陥」を防ぐプロセス改善を提案し, その改善活動を支援する「PM 改善ナビ」を開発現場に適用した結果を紹介する.

1. はじめに

本報告では, 「開発の"待ち", "遅れ", "欠陥"を防ぐプロセス改善」ついて論じ, 次に「PM 改善ナビを利用した"待ち", "遅れ", "欠陥"の改善活動」について(共に事例までを)論じる. 考察とまとめは, 2者をまとめて論じる.

前者の「開発の"待ち", "遅れ", "欠陥"を防ぐプロセス改善」では, 課題ばらし*1 の十分性と, プランニング*2 との整合性確保が, 改善のベース(土台)になることを論じたうえで, “欠陥”, “遅れ”, “待ち“ を防ぐ改善原則(3 種類) を紹介する.

*1) 技術的に曖昧な点や懸念事項を事前に明確にすること[5]. (補足:開発の阻害要因となる点はリスクと同じだが, 開発目標実現のために克服/解決が不可避となる点は, リスクの対極に位置する.)

*2) 課題を解決するための対策手順(段取りや作戦)を明確にすること

後者の「PM 改善ナビを利用した"待ち", "遅れ", "欠陥"の改善活動」では, 改善活動を支援する PM 改善ナビへ集約した知識(分析項目や定量指標)等について紹介する.

本報告の内容を正しく理解して頂くため, はじめに, WBSやレビュー, テスト等のソフトウェア開発管理で使われる用語が果たす役割について, 一般と本報告での違いを述べる. これは, (繰り返しになるが)本報告内容の理解のためであり, プロジェクト管理の定石やその効用を否定する意図はないことを予め明確にしておく.

開発工期はWBS(WorkBreakdownStracture)の総量と総マンパワーを基に算定する通念(一般論)が有るが, 実際にはWBS以外の要因の方が開発工期へ大きく影響する. 実際に要する期間は課題の大きさ(課題を解決して得られた付加価値の大きさ)と質(課題の新規性や複雑性)の累積に依存し, 課題の重さを把握する方法(課題の見積方法)が, 開発の遅延を抑制する決め手になる. 加えて, 開発品質はレビューとテストで如何に欠陥を除去するかによる, という通念(一般論)が有るが, 実際には欠陥の作り込みを抑制するプロセスが開発製品の品質(良し悪し)に大きく関わる. レビュー指摘欠陥やテスト検出バグの本質は後から分かった課題であり, 「課題ばらしの質の向上」の取り組みこそが開発(作込み)品質向上の決め手になる.

プロジェクトの管理はWBSをベースにした方法論が一般的である. しかし, 開発工期の支配要因ではないWBSを一般論の通りに管理しても, (前述した理由により)期待する開発遅延抑制の効果を得るのは難しい. 加えて, ソフトウェアの品質管理は「レビュー」と「テスト」をベースにした方法論が一般的である. しかし, 作込み品質向上の決め手とはならない「レビュー」と「テスト」を一般論の通

りに実施しても，（前述した理由により）期待する作込み品質向上の効果を得るのは難しい．また，欠陥の発生原因分析は，課題ばらしの振り返りと比べて属人性が高く(分析者への依存度が高い)，即時性と網羅性が十分得られないため，作込み品質向上の効果は限られる．

我々は，一般の知識体系を利用して改善に活用するのではなく，トヨタ開発方式の実践によって自前で得た経験や結果を体系化し，開発現場の改善活動で利用することにした．

2. 開発の"待ち"，"遅れ"，"欠陥"を防ぐプロセス改善

開発現場では，しばしば次のような問題に遭遇する．

- 実装後に不具合が多発した（開発終盤に起きると品質/納期の両面に悪影響）
- 予定した日程で機能を実現できなかった（特に新規性が高い機能の場合)
- 他部署から提供されたものが使用要件を満たさず，作業の着手が遅れた

トヨタ開発方式[1]では，1つ目の問題は"欠陥"，2つ目は"遅れ"，3つ目は"待ち"と捉え，3つとも開発の流れを阻害する要因と認識し，改善する対象としている．我々は "欠陥"，"遅れ"，"待ち"の問題に対し，トヨタ開発方式が提示する解決方法を開発現場の改善活動で利用することにした．

2.1. 問題の分析_その1

本章では"欠陥"，"遅れ"，"待ち"の問題の解決に向けて，分析した経過を述べる．

トヨタ開発方式は次の3つの改善原則を提示している．[1]

- 原則 1：問題を前段階で解決することにより，プロセス中の手戻りを無くす「"欠陥"の改善」．
- 原則 2：仕事や作業が足並みを揃えて進むように，調子を合わせる仕組みを作る「"遅れ"の改善」．
- 原則 3：横断的な活動を同期させ，必要な情報やものが，必要な時にやり取りされるようにする「"待ち"の改善」．

上述した3つの改善原則を参考に，弊社で"欠陥"，"遅れ"，"待ち"の問題を分析した結果を次に示す．

"欠陥"の分析：

ある製品（派生機種の開発）で，開発終盤に発生した不具合（手戻り）が課題ばらしの質（前段階のプロセス）に関係したかどうかを問い，調査した結果，大半が課題ばらしに起因する問題だった．そこで，課題ばらしの質と不具合の発生がどのように関係したかを問い，調査した結果，課題の抽出漏れが多い部分から不具合が多く発生していた．

"遅れ"の分析：

ある製品（新製品の開発）で，テーマ毎の遅延の大きさが見積りの精度に関連したかどうかを問い，調査した結果，新規性が高くなるにつれて工数の増加傾向が顕著だった．そこで，工数の増加が顕著になる理由を問い，調査した結果，機能追加と同様の課題ばらしと見積りを（新機能に対しても）一様に行なっていた．

"待ち"の分析：

ある製品（新ハードのサポート）でソフト評価作業の着手が遅れたことについて，ハード開発部署と事前に摺合せた内容を問い，調査した結果，使用条件について事前に打合せを行い，ソフトが使用する機能について合意していた．しかし，動作条件については，ハード開発側の考えとソフト開発側では差異が有った．

2.2. プロセス改善の土台

本章では，トヨタ開発方式を利用したプロセス改善のベース（土台）となる課題ばらしの十分性と，プランニングとの整合性について述べる．

“欠陥”，”遅れ”，”待ち”を防ぐ土台(その1)は，次に示すとおり課題ばらしの十分性を確保することである．

- 開発当初（～計画時）に，開発全体を通して 心配や不明な点を充分に洗い出し技術課題を明確化する（理由-> 必要な課題ばらしが漏れていると，後から分かった課題となって顕れる）．

“欠陥”，”遅れ”，”待ち”を防ぐ土台(その2)は，次に示すとおり課題と計画の整合性を確保することである．

- 抽出した課題の解決をベースにした作戦（優先順位の設定等）を立てる（理由-> 課題と計画の不整合は，課題ばらしの効果を減少させる）．

課題と計画の間の整合性確保は一見，出来て当たり前の様である．しかし，技術課題がチーム内で共有されていなかったり，課題の解決プランが担当者により明確化されていなかったり等の場合があり，実行するのは簡単ではない．

2.3. 問題の解決_その1

本章では，開発の"待ち"，"遅れ"，"欠陥"を防ぐプロセス改善について述べる．

トヨタ開発方式は"欠陥","遅れ","待ち"を改善するため，次の6つの行動原則を提示している．

⇒時間差解除による平準化[1]
-> 設計情報を一定の期間で（Monthly や Weekly で）分割して解除し，設計情報の取り出しや処理を行いやすくする

⇒組織横断活動の同期化[1]
-> 組織毎の入/出力と相互依存性をもとに，同時進行している活動がうまく統合するように調整する

⇒顧客価値の探究と実現[1]
-> 顧客価値を創り込み実現するように，要件を洗練し網羅する

⇒セットベース開発[1]
-> 多角的な視点から技術課題を検討し設計の最適解を導出する
※:弊社では，課題ばらしの質が一定以上に達した状態をセットベース化された状態とした

⇒知識の再利用[1]
-> 知識を形式化して再利用し，技術課題の抽出力を向上する
※:弊社では，課題ばらしの再利用を知識の再利用とした

⇒LAMDA サイクル*3 の利用[6]
*3)PDCA サイクルを開発向けに発展させたもの
-> チームで開発と改善を同時に行い，問題を着実に解決する

前述した改善原則(項 2.1)の実行は，上述した6種の行動原則(トヨタ開発方式の知見)から，問題解決に必要なものを選択する事で実現する．6種の行動原則は課題ばらしの質向上の取組みの中で具体化される．

弊社で"欠陥"，"遅れ"，"待ち"の問題の改善方法を検討した結果（行動原則の選択を含む）を次に示す．

"欠陥"の改善:
課題ばらしの質を一定水準へ改善すること（セットベース化）で，開発終盤の不具合を減らせるか確認する．加えて，どの程度の水準へ改善すれば不具合が減るのか，その水準値も把握する．【セットベース開発】

"遅れ"の改善:
新規性の高いテーマの見積りをチームで改善し，見積りと実際の課題の大きさの差を縮小して遅れを抑制する．【LAMDA サイクルの利用】

"待ち"の改善:
ソフト開発側で期待値を明確にしてハード開発側に伝え(期待値との差異を確認または生じないようにし)，部署間のマイルストンを設定する．【組織横断活動の同期化】加えて，テストを細分化してハード開発と並行して進められる様にする（ハードの機能を段階的に提供してもらい，ソフトの評価を逐次進められる様にする)．【時間差解除による平準化】

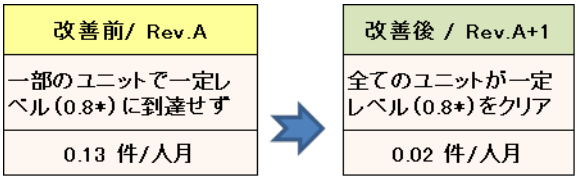
2.4. 改善事例

本章では，開発の“欠陥”，“遅れ”，“待ち“ を改善した事例を紹介する．

最初に，課題ばらしの質を一定レベル確保し，開発終盤での不具合修正を抑制した事例（“欠陥”の改善例）を紹介する．品質要求水準が高い製品開発で課題ばらしの質確保に取り組んだ結果，品質が向上した．この活動で使用した改善指標と結果（改善効果）を以下に示す．

表 1 改善指標と改善効果

	測定項目
指標1.	課題ばらしの質 ＝事前にばらせた課題 /（事前にばらせた課題＋後から分かった課題）
指標2.	パフォーマンス<品質> ＝ 最終マイルストンで行った不具合修正件数 / 開発工数[人月]



90 人月の製品開発　　45 人月の製品開発

上述した改善について，経過を以下に示す．

活動にあたり，初めに課題ばらしの質を測定するための計測基準を設け（全チームで統一），あるレビジョン（Rev.A）から計測を開始した．結果を分析したところ，最終マイルストンで行った不具合修正が特定ユニットに集中し，このユニットの課題ばらしの質は一定レベルに届いていなかった．一方，課題ばらしの質が一定レベル確保されていた他のユニットでは，最終マイルストンでの不具合修正は殆ど無かった．不具合が多発した特定ユニットの振返り（分析）記録では，約8割が課題ばらしの問題（事前抽出が可能）となっていたことに加え，改善案（振返り結果）を実行することで，他のユニットと同一水準へ改善できることが分かった．　不具合修正が多発した特定ユニットの開発チームは，次のレビジョン（Rev.A+1）で改善策を実施した．結果，課題ばらしの質は一定レベルを超え，品質が大幅に向上した（最終マイルストンでの不具合修正は無くなった）．製品全体においては最終マイルストンで行った不具合修正は1件のみとなり，品質が大

幅に向上した.

上記の事例は, 課題ばらしの質が確保されると後から分かる課題が減り, 不具合が発生しにくくなることを裏付けた. 加えて, "欠陥"の改善に関しては, 6種の行動原則から, セットベース開発を選択し, 実施することが有効であることを示した.

次に, 課題の見積スキルの差をチームで補い, 見積精度のばらつきを低減した事例(“遅れ”の改善例)を紹介する. 新規性が高い製品開発で見積精度のばらつき低減に取り組んだ結果, 工期遅れを低減した(開発規模は約180人月, アジャイル方式により月1回のリリース). この活動で行われた改善策(4 点)を以下に示す.

① 全ての課題を対象に新規性を確認(大/中/小)
→新規性“大”(従来製品には無い機能等)のテーマについて②以降の対策を実施
② 担当者の課題見積り結果を関係者で再確認
③ 数バッファを持った計画を策定(チームで共有)
→テーマ毎に 1W 又は 2W のバッファを設定(2Wを超える場合は次のリリースへ繰り越す)
④ 積工数と実績工数の差を確認し分析
→使用したバッファの中身を分析しバッファ使用率を下げる改善策を検討

上記の改善(見積精度ばらつきの低減)を行った結果, 次のとおり, ほぼ期日通りにリリースできるようになった.

リリース期日達成率[%]*4

改善する前		改善後
0%	⇒	86%

***4) リリース期日達成率**
＝期日達成リリース数 / リリース数 ×100[%]

上記の事例は, 新規性が高いテーマの見積り精度が確保されると, 課題の大きさと見積りの差が縮小し, 遅れが生じにくくなることを裏付けた. 加えて, "遅れ"の改善に関しては, 6種の行動原則から, LAMDA サイクルの利用を選択し, 実施することが有効であることを示した.

最後に, 情報やものの必要時期と内容(機能や検証の網羅度)について提供部署の合意を(事前に)取り, 開発中に生じた変化にも適切に対応した事例(“待ち”の改善例)を紹介する. 評価用のハードウェアが提供されたが(完成度が低いため), 予定通りに作業に着手できなかった問題を振返り*5, 次の開発で改善策を実施した結果, 待つ状態を最小化した(開発規模は 10 人月位から 100人月迄の範囲内). この活動で行われた振返りの結果(改善策)と効果を以下に示す.

*5)振り返り[5] → Y:やったこと, W:わかったこと, T:つぎにやること

Y:“動作実績のあるデバイス実測評価” というマイルストンが遅延した.

W:ハード開発から”キャプチャできる” という情報があったが, ソフトウェアテストに耐える状態では無かった.

T:ソフト開発側の期待値を明確にした部署間マイルストンを設定する.
例)“動作実績のあるデバイス実測において, キャプチャが 100%動作可能なハード状態で, 評価を開始する”.
加えてテストを細分化し, ハード開発と並行して進むソフト開発側の対応に柔軟性を持たせる.

次期開発で上記の改善を行った結果, ハード開発遅れに対してソフト開発側の影響を最小化できた.

上記の事例は, ハード開発とソフト開発が並行して進む際, ソフト開発側でマイルストン(部署間および部署内)を設定し調整すると, 着手遅れを抑制できることを裏付けた. 加えて, "待ち"の改善に関しては, 6種の行動原則から, 組織横断活動の同期化と時間差解除による平準化を選択し, 実施することが有効であることを示した.

3. PM 改善ナビを利用した"待ち", "遅れ", "欠陥"の改善活動

我々は, 開発の“欠陥”, ”遅れ”, ”待ち” の改善に取り組んだアウトプットとして, 改善の実践で得た経験や結果を体系化し, (改善の参考情報を)参照しやすい形で提供する Tool を作成した. そして, プロジェクト管理において改善課題と解決策を検討する際の道標になってほしいという願いをこめて,「PM(Project Management)改善ナビ」と名付けた.

本章では, 開発の“欠陥”, ”遅れ”, ”待ち” の改善活動で得た経験をトヨタ開発方式の知見に基いて体系化し Tool 化(PM 改善ナビを作成)した経過, および, PM 改善ナビ(以降 3.章の中で”ナビ”と略記する)を開発現場で試用した結果を述べる.

3.1. 問題の分析_その2

本章では"欠陥", "遅れ", "待ち"の改善の普及に向けて, 課題を分析した経過を述べる.

前章で述べた通り, トヨタ開発方式は"欠陥","遅れ","

待ち"を改善する効果があることを確認した. 但し, これまで行ってきた開発現場に張付く方法では, 開発全体へ改善の知見を普及することが難しい. 普及していくためには改善活動で得た知見を参照しやすい形で提供することが不可欠である. 改善の知見を参照する際に欠かせないことは次の2点である.

・"欠陥", "遅れ", "待ち"の改善とは何か, が簡潔に示される
・"欠陥", "遅れ", "待ち"に関する現状分析と改善策の検討が効果的に行える

我々は "欠陥", "遅れ", "待ち"の改善に関する知見を開発全体へ普及するため, 新たな取り組みを行った.

3.2. PM 改善ナビの目標

本章では, ”ナビ”の作成経過と利用モデルについて述べる.

トヨタ開発方式の行動原則の一つに「設計の時間差解除」というのがある. これは, 機能を切り出して逐次開発を進める工法であるが, 内容的には Agile 方式と符合する. また, 切り出した開発毎(終了時)に計画(目標)と実績の差を精査し次の(切り出した開発の)計画(目標) を設定する点についても Agile 方式と符合する. 我々は, Agile 方式で開発している組織ではトヨタ開発方式を基にした知見を利用しやすい環境にあると考え, PM 改善ナビは Agile 方式の開発向けに作ることにした.

“ナビ”の試用にあたり, 次の利用モデル(2点)を設定した.

利用(推奨)モデル:

a. 課題ばらしの質の確保や向上について, 強化が必要なポイントを正しく導出できるかどうか確認する.
b. 開発体制や基盤等が変わる際(新規の製品開発等), 仕組みの内容に(改善の土台を構築する点で)漏れやがないことを確認する.

3.3. 問題の解決策_その2

本章では, ”ナビ”を使う前提条件と概要を述べる.

“ナビ”を利用する際, 課題ばらしと振返りの反復の実施が前提となる. 課題ばらしと振返りの反復は日常の改善活動の基本でもある. また前述した通り, 課題ばらしの質向上の取組みは, "欠陥", "遅れ", "待ち"を改善する6種の行動原則と深く関わる. 取組み方法は次のようになる.

"欠陥"の改善では, レビューで指摘された課題抽出の漏れ等を振り返り, 後から分かる課題を減らす改善策を立案する(日常). "遅れ"の改善では, 課題解決の後続マイルストンへの繰越し等を振り返り, 見積精度を向上する改善策を立案する(日常). 改善策を立案する際の現状分析は, “ナビ”が表示する分析項目(項 3.3 で紹介)を参照して行う. 開発全体での取組みは, 課題の抽出や見積の精度に関する実績を定量的に把握し(項 3.3 で紹介), 改善目標と課題を設定する(開発終了時).

続いて, ”ナビ”の概要を以下に述べる.

開発者が"欠陥", "遅れ", "待ち"に関する現状を分析し改善策を検討する際に, “ナビ”にはどの様な機能が求められるかを検討した結果, 次の3点を抽出した.

・開発作業を振り返る際に利用できること
・問題分析の際に表示する内容はトヨタ開発方式の6種の行動原則に基づいていること
・開発全体を振り返る際に使う定量指標が備わること

1点目は, 改善領域の選択と分析アクションで対応し, 2点目は”ナビ”の分析構造で対応した. そして, 3点目は, 定量指標の利用で対応した. これらの”ナビ”の機能について, 概要を以下に述べる.

1)改善領域の選択:

振返りのきっかけを表示し, ユーザは改善する領域を選択する.

【選択】		【振返りのきっかけ(例)】
□欠陥の振返り	←	レビュー時に課題の抽出漏れを指摘された.
□遅れの振返り	←	課題の解決を後続のイテレーションへ繰り越した.
□待ちの振返り	←	1Wを超える着手遅れが発生した.

2)分析対象アクション:

"欠陥", "遅れ", "待ち"の画面別に分析対象アクションを表示する.

【画面】		【分析対象アクション】
"欠陥"の改善	→	課題ばらし(技術的に不明や心配なことを事前に明確化すること)
"遅れ"の改善	→	課題の見積(課題の重さを把握し, 重さに応じた工数を設定すること)
"待ち"の改善	→	同期化作業(必要な時に必要なものがやりとりされるように, 関連する機能や作業と同期

を取ること)

3) ”ナビ”の分析構造:

"欠陥", "遅れ", "待ち" を分析しやすくする様に, 4つのガイドワード*6(①～④)に基づくメッセージを表示する.

*6) [7]

①アクション(全体または一部)が実行されない
⇒作業項目(課題ばらしの実施項目)に抜けがないか確認. 1 つの開発が複数の小さな開発の集合体となるように切り出されていることが前提「時間差解除による平準化」

②正しくないアクションが実行される
⇒作業手順が顧客要求の実現から逸脱していないか確認「顧客価値の実現」

③アクションのタイミングや順序が正しくない
⇒作業する時期が全体計画から逸脱していないか確認「組織横断活動の同期化」

④アクションを早く止め過ぎるか長く続け過ぎる
⇒作業のペースが早すぎまたは遅すぎないか確認(分析や検討を早く止め過ぎると, 予定期間で作業が終了しなかったり, 作業終了後に不具合が発生する可能性が高くなる)「セットベース開発」

4)定量指標の利用:

課題ばらしの質とパフォーマンスを見える化する指標(数値化方法と目安値*7)を表示する.

*7) 数年間・数十のソフト開発プロジェクトの実績(確認/調査結果)を基に目安値を設定

"欠陥"の改善【手戻りを減らす】

課題抽出率＝事前に抽出した課題 / (事前に抽出した課題＋後から分かった課題)
⇒目安は, 0.85

欠陥発生率＝作業終了後に他部署から指摘された不具合件数 / 開発規模[人月]
⇒目安は, 0.1～0.2 以下

"遅れ"の改善【計画精度を向上】

課題解決率＝期間内に解決した課題 / (期間内に解決した課題＋繰り越した課題)
⇒目安は, 0.85

遅延発生率＝遅延発生イテレーション数 / イテレーション総数
目安は, 0.1 以下

"待ち"の改善【予定通りに着手】

同期化設定率＝同期化点の設定数 / 同期化が必要な点の数
⇒目安は, 1.0

同期化成功率＝対策が成功した同期化点の数 / 対策を実施した同期化点の数
⇒目安は, 0.85 以上

3.4. 適用事例

本章では, ”ナビ”の適用例(試用結果)を述べる.

部門A:開発基盤が整備され(部門内の改善手順書有り), 改善活動のマンネリ防止が課題
効果 ⇒ 開発現場で使われている改善手順書に不足している点(または無い点)を補完
※:改善手順書に, 定量指標を追加(2点:課題ばらしの質の評価, パフォーマンスの評価)

部門B:新たな開発基盤を整備中(SCRUM 導入)で, 振返り(SPRINT 毎)の定着が課題
効果 ⇒ 整備中(SCRUM)の基盤上で, 振返りの基準(タイミングと対象)を明確化
※:開発現場の規定に, 基準を追加(2点:振返りは SPRINT 毎に実施, 課題抽出漏れのレビュー指摘は振返り必須)

部門C:開発製品が異なるチームが合流し, リーダやメンバの早期立上げが課題
効果 ⇒ 新たに加わった開発チームが, 開発基盤の意義を理解し実行する為の課題(必須部分)を明確化
※:チームの改善課題を設定(2点:課題や見積結果はチームで共有, 振返り結果は極力汎化する)

部門Aでは課題ばらしの質の評価指標を使うようになり, ”ナビ”が課題ばらしの質の改善に寄与することを確認した. 部門Bでは SPRINT 毎に振返りを行うことが組織のルール化となり, ”ナビ”が改善の土台を構築することに寄与することを確認した.

上記の試用結果から, “ナビ”は目標(利用モデル)に

対し有効であることを確認した.

4. まとめ

開発工期と品質の改善に取り組む際, 開発の流れを阻害する"待ち", "遅れ", "欠陥"を防止していくことが課題となる. 開発の"待ち", "遅れ", "欠陥"を防ぐには, 3つの改善原則("欠陥"の改善, "遅れ"の改善, "待ち"の改善)と, これを実現するための6種の行動原則(選択と実施)が有効である.

この度, 開発作業を振り返る(現状分析する)際に必要な知見を“ナビ”へ組み込んだ. 今回組み込んだ知見は, "欠陥", "遅れ", "待ち"の改善活動(開発終了まで)で得た知識である. 今後は, “ナビ”へ収録した改善の知見を「開発終了後に発生した欠陥の防止」へ拡充していく必要がある.

5. 今後の課題

今後, "欠陥","遅れ","待ち"の改善を進める上での課題は何かを問い, 開発が置かれている状況を調査した. 結果, 労働時間短縮の流れの中で保守工数の割合が増加していた(理由は開発終了後の欠陥の減少よりも開発稼働時間の減少の方が顕著なため). 弊社での保守作業は開発作業とのマルチタスクになる場合が多く, 保守工数の割合増加は開発スループットの低下要因となる. そこで, 我々はライフサイクル全体で(開発終了後の不具合発生低減を含めて) "欠陥","遅れ","待ち"の改善に取り組むことを新たな課題とした.

開発終了後の欠陥を減少させるために必要なことを問い, 調査した結果, 大半は課題ばらしに起因する問題であった(レビュー・テストで検出できず市場へ流出). 加えて開発終了後に発見される不具合は複雑度が高く, (開発終了後の不具合修正は全て改善策の報告対象となっているが)前段階での解決が容易でないことを確認した. とはいえ, 分析・検討時に抽出が漏れてしまった課題は, レビュー・テスト時には捕捉がさらに難しい.

現在, 開発終了後の不具合を低減するための課題ばらしについて検討を進めている. この改善策を導き出すため, 取組みはトヨタ開発方式が提示する問題解決方法(6種の行動原則)の中から, 次の2つを選択した.

・顧客価値の探究と実現

・知識の再利用

上記の取組みにより, 開発終了後の不具合を低減する課題ばらし方法を明確化していきたい.

参考文献

[1] James M.Morgan and Jeffrey K.Liker:
The TOYOTA Product Development System
トヨタ製品開発システム(翻訳), 日経BP社, 2007

[2] 比嘉定彦他:SQiP2013_報文_B3-1 トヨタ開発方式の利用によるソフト開発の QCD 向上活動, (財)日本科学技術連盟, 2013

[3] 比嘉定彦他:SQiP2014_報文_B3-1 トヨタ開発方式の深堀りによるソフト開発の QCD 向上活動, (財)日本科学技術連盟, 2014

[4] 比嘉定彦他:SQiP2015_報文_B3-1 トヨタ開発方式の包括的利用にによるソフト開発の QCD 向上活動, (財)日本科学技術連盟, 2015

[5] 中村素子, 勝田博明:技術者・エンジニアの知的生産性向上, 日本能率協会マネジメントセンター出版, 2009

[6] 稲垣公夫:開発戦略は「意思決定」を遅らせろ, 中経出版, 2012

[7] ET2015/IoT2015 併催:IPA セミナー
http://www.ipa.go.jp/sec/seminar/20151119.html
IPA セミナー【第 4 部】
～組込みシステムの安全解析と障害原因診断分析の統合アプローチ／STAMP～

ゲーミフィケーションを用いた探索的テストの効果報告

根本 紀之
東京エレクトロン
noriyuki.nemoto@tel.com

1. 要旨

テスト仕様書ベースのテストが好きな開発者は少ない．もちろん例外はあるだろうが，一般的には少ないと言って過言ではないであろう．理由は創造的な活動ではない，テスト仕様書の作成に時間がかかる，など様々である．

本報告では，テスト仕様書ベースのテストの一部を，探索的テストに変え，さらにゲーミフィケーションを取り入れることで，バグを探すこと自体が面白くなり，通常用いているテスト仕様書ベースのテストより多くのバグを発見することができたという結果とその効果を報告する．さらにはテスト実行前に戦略立案，テスト実行後にふりかえりを行うことで，バグの効果的に見つける方法を共有する取り組みを紹介する．

2. 背景

『テストは単純作業で楽しくない．どちらかというと面倒で辛い作業』これが一般的な認識に近いであろう．筆者の経験でも開発者はテストフェーズに入ると，プログラミング時と比較して，モチベーションが下がるエンジニアが多いと感じていた．

図 1 は筆者が描いたテスト仕様書ベースのテストの問題構造図である．『テストフェーズは楽しくないので，モチベーションでダウンする』という問題が，『バグを積極的に見つけようとしない』マインドを引き起こし，最終的にリリース後のバグに繋がっているではないかという仮説に行きついた．

もう一つ着目したのは『自分が経験していないバグの出し方は良く分からない』という問題である．BTS を用いることでそのバグの現象や再現方法は共有できるが，筆者の開発経験の中ではバグを出す考え方は共有されることが少なかった．

これらの問題を解決することで，限られた時間の中で開発者が楽しく効率的にバグを見つけることができると考えた．

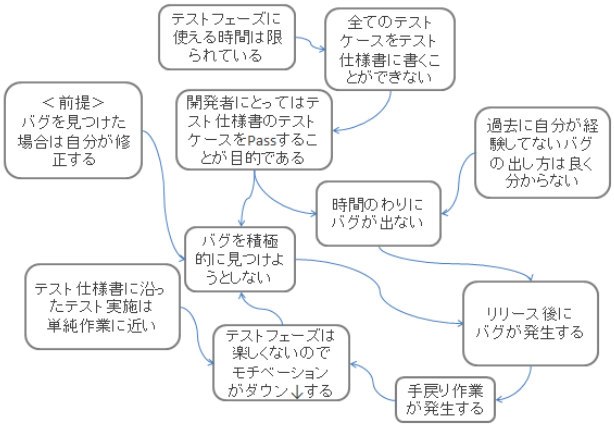


図 1 テスト仕様書ベースのテストの問題構造図

3. 目的

楽しくないと思われているテストにゲーミフィケーションを用いることでエンジニアの競争心を刺激し，モチベーションアップ，バグ検出効率向上に有効であることを確認すると共に，テスト戦略とバグの出し方を共有するふりかえりの効果を確認することを目的とする．

4. ゲーミフィケーションとは？

ゲーミフィケーションという言葉は「日常生活の様々な要素をゲームの形にする」という「ゲーム化（Gamefy）」から派生し，2010 年頃から使われはじめた[1] ．.ゲーム業界の中で独自に培われていたユーザを楽しませ，俗に言われる『ハマる』という状態を作りだすためのノウハウを教育や販促や人事評価など非ゲームのコンテキストで使用するのがゲーミフィケーションである．ゲーミフィケーションの基本的な要素は PBL と言われ，それぞれ P：ポイント，B：バッジ，L リーダーボードであり，最近は他の参加者とのチャットやチームでの取り組みなどソーシャ

ルな要素も重要視されている．ゲーミフィケーションという言葉自体は新しいものであるが，実は昔から身近な生活に取り入れられている．たとえば，ラジオ体操のスタンプカードなどもゲーミフィケーションの一つであるが，ラジオ体操自体の魅力は低くても，スタンプを押してもらうことに楽しみを見出し，夏休みの間，一回も休まず通った記憶を持つ方も多いのではないだろうか．

ソフトウェアテストに関連するゲーミフィケーションの論文としては Microsoft の Language Quality Game という事例が存在している．これは多言語対応されている Microsoft のダイアログボックスの誤記や意味の通じないメッセージを見つける作業をゲーム化した事例であり，参加者 4500 人，チェックしたダイアログボックスは 50 万以上，報告されたバクが 6700 件という結果が報告されている[2]．

5. 探索的テストとは？

James bachによると，「探索的テストは，学習，テスト設計，テスト実行を並行して実施するものである．」とされ [3]，従来のテスト仕様書ベースのテストであるスクリプトテストと区別される．探索的テストはアジャイル開発との親和性も高いことから，北米を中心とする海外ではメジャーなテスト手法となってきている[4]．

探索的テストはドキュメントの作成とメンテナンスのコストが少なくて済み，またテスト実施中に対象ソフトウェアの動作をフィードバックすることで怪しいところを重点的に攻めることができることから，一般的には費用対効果が高いと言われている．一方で，バグを出す考え方やノウハウは属人化する傾向が高く，探索的テストの課題の一つとなっている．

6. 手法

ゲーミフィケーションを用いた探索的テストの対象となる機能は，探索しやすいようにGUIを含み，厚くテストを実施する必要があるとチームで合意した機能とした．またこの機能は実験用の機能ではなく，実際に製品として開発した機能である．参加したエンジニアは 5 年目～15 年目の中堅エンジニア～ベテランエンジニアが中心であり，それぞれ設計，実装，テストと一通りの開発経験がある．テストは期間を置いて2回実施し，1回目は個人戦，2 回目はチーム戦である．それぞれの特徴を表 1 に示す．

個人戦では，テスト実施の時間は 1 時間のタイムボックスとし，テスト対象は参加者全員同じ機能を対象とした．同じ時間，同じ機能をテストすることで，公平性が生まれる．バグは見つけた開発者がその場で現象を共有ファイルに書き込んでいく．バグは重要度の高い順にAランク，Bランク，Cランクとポイントを規定し，テスト後のふりかえりで，ランク付けを行い，個人のポイントを決定した．また，ふりかえり時にはバグの現象だけではなく，そのバグをどのように出したかという暗黙知を共有するための質問を積極的に実施した．ふりかえりの時間は 1 時間程度とし，最後に個人 TOP を表彰した．

チーム戦も同じ時間に同じ対象についてテストを実施するという基本のルールは個人戦と同じである．チーム編成では，公平を期すため少なくともチームに一人はその機能の知見者を配置した．チーム戦で新たに付け加えた施策は，テスト前の戦略立案とリーダーボードの二つである．戦略立案は今回のイテレーションで開発された案件や過去のバグを確認し，テストで狙うべき箇所とチームでの役割分担を計画する．このアプローチは探索的テストから派生したセッションベースドテスト[5]に近いと考えている．

Web アプリであるリーダーボード（図 2）はテスト実施中に他のチームが発見したバグの件数がリアルタイムで分かるようになっている．この情報はあくまで件数であり，ランクのポイントが考慮された現在の順位は分からない仕掛けにしている．

表1 個人戦とチーム戦の違い

タイプ	個人戦	チーム戦
単位	1 人	3 人 1 チーム
参加人数	5 名	9 名 (3 チーム)
テスト対象機能	AAA 機能	BBB 機能
テスト前の戦略立案	なし	あり
テスト実施時間	1 時間	
テスト実施中の リーダーボード	なし	あり
テスト後のふりかえり	あり	
ポイント配分	A ランク ：5 ポイント B ランク ：3 ポイント C ランク ：1 ポイント	
賞	個人 TOP	個人 TOP チーム TOP

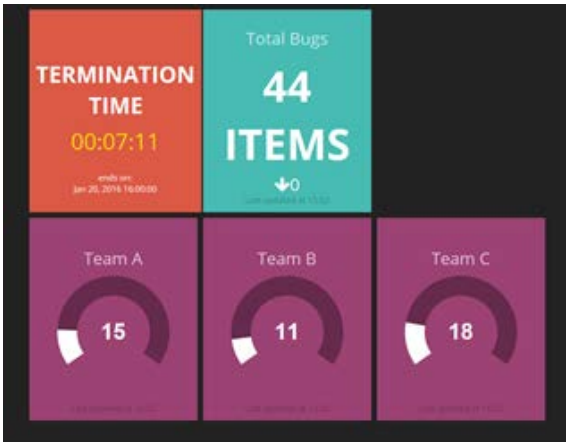


図 2 テスト実施中のリーダーボード

7. 結果

7.1. モチベーションアップ

テスト実施後に，「テスト仕様書に沿ったテストと比べて楽しかったですか？」というアンケートを取った．個人戦のアンケート結果を図 3 に，チーム戦のアンケート結果を図 4 に示す．チーム戦後のアンケートでは変わらないという人もいるが，平均的には楽しかった人が多くなっているのが見てとれる．

ゲーミフィケーションを用いた探索的テストの仕組みにより，モチベーションの低下を抑制できたと考えてよいであろう．筆者自身も実際にテスト実施の間は，他の参加者に負けたくない！高得点に繋がるバグを見つけたい！という思いでテストを実施していた．

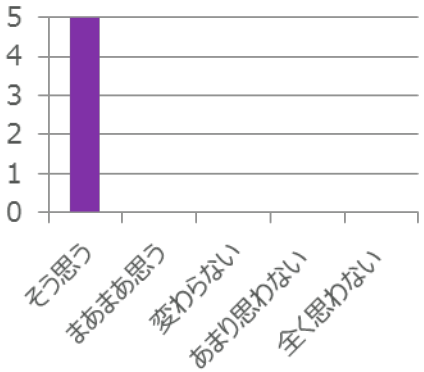


図 3 テスト仕様書に沿ったテストと比べて楽しかったですか？のアンケート結果（個人戦）

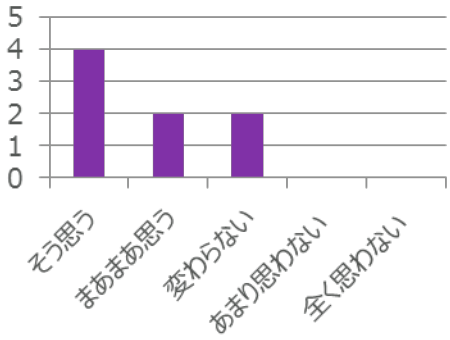


図 4 テスト仕様書に沿ったテストと比べて楽しかったですか？のアンケート結果（チーム戦）

7.2. バグ検出効率の向上

個人戦におけるバグ検出のデータを表 2 に，チーム戦におけるバグ検出データを表 3 へ示す．単位時間あたり個人戦では 5 件，チーム戦では 6.4 件のバグを検出した．

発見されたバグには既知のバグとの重複や，同じ対象をそれぞれの参加者がテストするため，参加者同士でのバグの重複がある．

表 2 バグ検出データ（個人戦）

総発見バグ数	25
一人当たり平均	5
有効発見バグ数	25
A ランク	1
B ランク	5
C ランク	17
確認待ち	2

表 3 バグ検出データ（チーム戦）

総発見バグ数	58
一人当たり平均	6.4
有効発見バグ数	22
A ランク	0
B ランク	5
C ランク	16
確認待ち	1

7.3. バグの効果的な出し方の共有

「ふりかえりの共有のときに新しい発見がありましたか？」というアンケートを取った．個人戦のアンケート結果を図 5 へ，チーム戦のアンケート結果を図 6 へ示す．

このアンケート結果からは，ふりかえりでは何らかの発見があったことが読み取れる．実際のふりかえりの中では，バグをどうやって出したか，現状の仕様がどうやってできあがったかという経緯の説明，それぞれの開発者が思っているバグの傾向など通常共有されない暗黙知が次々と出ていた．

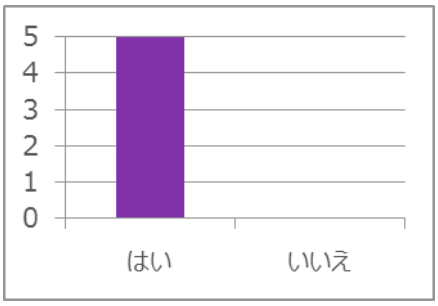


図 5 ふりかえりのときに新しい発見がありましたか？のアンケート結果（個人戦）

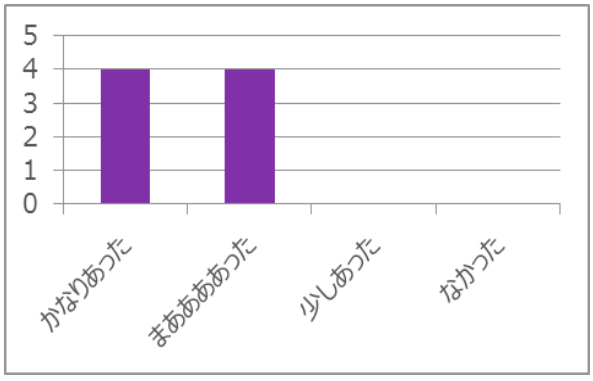


図 6 ふりかえりのときに新しい発見がありましたか？のアンケート結果（チーム戦）

8. 考察

8.1. 課題と解決策

それぞれの課題に対して効果が高かった解決策をマッピングしたものを図 7 に示す．今回は探索的テスト，ゲーミフィケーションの 2 つの施策が上手く絡まりあって，テスト実行のモチベーションアップ，短時間での大量のバグを検知できたという良い結果に結びついたと考える．

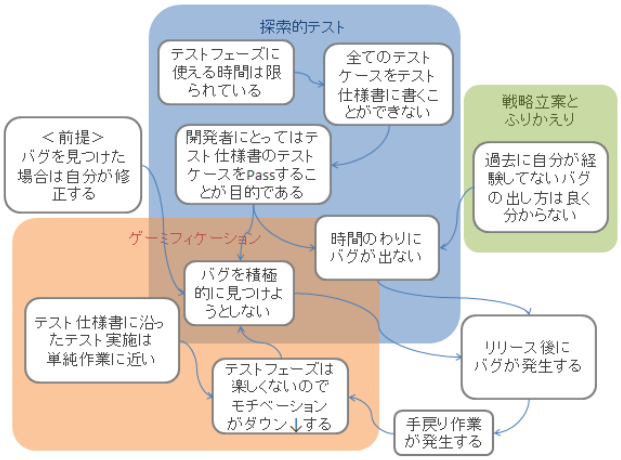


図 7 問題点と解決策のマッピング

8.2. ゲーミフィケーションについて

今回のゲーミフィケーションを用いた探索的テストではどのようなことが起こっていたのかを考察する．テスト仕様書に沿ったテストに比べ，時間が経つのが早かったですか？という，アンケートの個人戦の結果を図 8，チーム戦の結果を図 9 に示す．全員ではないが，半数以上はそう思う，まあまあ思うと回答している．この結果から，時間が早いと感じた開発者は集中した状態であり，ゲーミフィケーションの特徴の一つである『ハマる』という状態，つまり一般的に言われるフロー状態に入っていたのではないかと考えられる．

前述したとおりテスト実行の時間は 1 時間であるが，終了時には一気に疲れがくるとアンケートの感想に書かれていた．この感想もテスト仕様書ベースのテストより集中してテスト実行していたことを裏付けている．

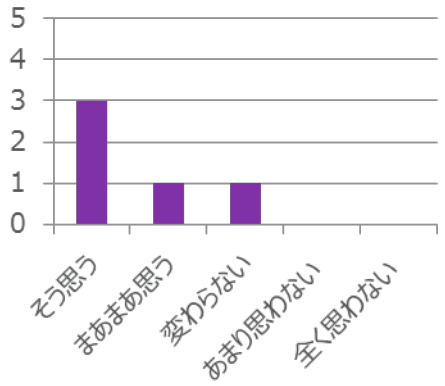


図 8 テスト仕様書に沿ったテストに比べて，時間が経つのが早かったですか？のアンケート結果（個人戦）

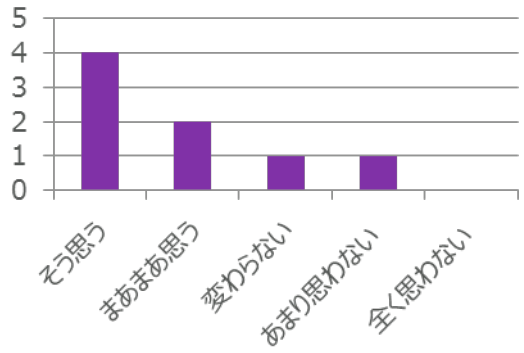


図 9 テスト仕様書に沿ったテストに比べて，時間が経つのが早かったですか？のアンケート結果（チーム戦）

次に『ハマる』状態を作り出す主要因として，3 つの要素を考えた．一つは，同じ時間，同じ機能を対象として，条件を公平にしたことで，純粋にバグを出す能力に焦点を当てたことである．仮に別の機能で同じようなことを実施しても，バグが多い機能に当たったからポイントを取ったという見方もできるであろう．

もう一つは開発中の機能であるため，適度なバグが存在したということである．もちろんプロダクトとしては褒められる状態ではない．しかし自分でバグをどうやったら出せるか考え，その操作することで，実際にバグを発見し，ポイントが手に入るという成功体験を繰り返すことが，楽しいという感覚に繋がっていくと考えられる．仮に，対象となる機能の品質が高く，1 時間で 1 件もバグを見つけることができない場合には，楽しかったという結果にはつながらない可能性が高い．

最後の一つはバグを見つけると褒められる仕組みにしたことである．本来，社内でバグを検出するということは市場にバグを流出させる前に止めることであり，喜ばしいことである．しかし，リリース直前などでは，「なんで今バグを出すんだ！」などと言われた経験がある人も多いのではないだろうか．今回はランクが高いバグを出すことでポイントを手に入れることができるというルールにしたため，バグを出すことは良いことであるという共通認識になり，積極的にバグを見つけるモチベーションにつながったと考えられる．

8.3. 戦略立案とふりかえりについて

チーム戦では戦略立案を取り入れたが，この戦略には，簡易ではあるがテスト分析，テスト設計の要素が含まれている．対象機能の知見を持っている開発者を必ず一人以上チームに配置することにより，その機能の特徴や過去のバグなどを共有することができた．ただし，戦略に沿ってテストをするだけになってしまうと，ソフトウェアを動かしたときの違和感などのフィードバックを見逃す可能性があるので，その点は気を付けて進める必要がある．

ふりかえりでは，バグの現象とその発見に至った経緯の共有に重点を置いた．同じ機能に対してテストしたからこそ，自分には無いテスト観点や，考え方の違いを受け入れやすかったと考えられる．異なる機能をテストしていた場合ではコンテキストが共有されていないため，詳しく説明されてもイメージが湧かない可能性が高い．

したがって，この戦略立案とふりかえりの両方の取り組みが，暗黙知を共有する良い仕組みであったと考えられる．この活動は探索的テストではなくとも使えるため，テスト仕様書ベースのテストでも実施することは可能である．

8.4. マイクロソフトの取り組みとの差異

マイクロソフトのテスト対象は膨大にあるダイアログボックスである．作業としては単純作業であり，そこにゲーミフィケーションを使うことで大人数のエンジニアが自発的に参加し，大量のバグを検出することができた．

一方，本報告で取り組んだテスト対象は特定の機能であり，実施した探索的テストも単純作業ではない．しかし，今回の取り組みとアンケートの結果から創造的な探索的テストにもゲーム性を持たせることは可能であることが分かった．テストの時間や対象を同じにするなど，公平な条件にすることで知的なゲームとなり得るであろう．

9. 課題

9.1. ゲームバランスの調整

ゲームバランスの調整として，二つの問題がある．まずはランクによるポイントの変更を考える必要がある．最終的な目的は A ランクである重要バグを検出することであるため，A ランク発見時のポイントを引き上げる必要がある．ランク毎の数の比率も考慮に入れると，ゲーム的にも一発逆転が可能な 20 ポイント程度が適切だと考えている．

もう一つは既知の不具合に対するポイントの検討である．例えば既知の不具合に対するポイントを 0 ポイントにした場合は，知っている不具合でポイントを稼げてしまうというチートを防ぐと共に，高得点を出すためには過去のバグを確認するという行動を誘発することができる．

9.2. ゲーム性を高めるUIの構築

今回は各チームの状況を見ることができる Web アプリのリーダーボードを用意した．しかし，さらにゲーム性を高めるアイディアを実装することで，エンジニアは楽しくバグを探すことができる．まだアイディアレベルであるが，以下の施策を候補として考えている．

・A ランクのバグを見つけた場合は，他の参加者へ通知する

・時間 10 分前になるとデータが見えなくなる

9.3. バグ報告の簡易化

バグ報告には現象と共に画面キャプチャーなどが必要である．その作業が簡単にでき，バグを見つけ出す思考が止まらないような仕組みを作る必要がある．最終的には検出したバグを精査後に，BTS として使用している Redmine に登録するが，この作業も開発者の負担となっているため，ワンクリックで Redmine に登録できる仕組みも考える必要がある．

9.4. 重複バグへの対策

時間と対象を同一にして複数人でバグ検出をしているため，複数の重複したバグが発見される可能性がある．その点に関してはプロダクトの成熟度合いなどを考慮に入れて，ゲーム実施の人数を絞ったり，テスト対象を変えたりする必要がでてくるであろう．ただし，テスト対象を変えると公平さが失われる可能性がある．

9.5. 通常の開発プロセスへの適応

時間と対象を同一にすることがゲームの公平性を保つために重要であるが，この状況を通常の開発プロセスに適応するのは前述した費用対効果の面からも難しいことが予想される．

アイディアレベルであるが，以下の施策を実施することで，一定の公平性を保ちながら開発プロセスへ適応することが可能になると考えている．

・対象をある開発フェーズで入ったすべての機能とし，どの機能を探索するかも戦略の一つとする．

・チームがどの機能を探索するかを抽選とする．

10. 謝辞

今回の報告において，初めての手法に面白そうだと理解を示し，一緒に取り組んでくれたチームメンバーである的川建史様，藤村浩様，大谷陽介様，小川裕亮様に感謝いたします．またリーダーボードを作成してくれた平井有希様に感謝いたします．また忙しい中，レビューを実施してくれた小楠聡美様に感謝いたします．

参考文献

[1] Wikipedia https://ja.wikipedia.org/wiki/ゲーミフィケーション

[2] Kevin Werbach, 渡部典子訳, 2013, ”ウォートン・スクール ゲーミフィケーション集中講義”, CCC メディアハウス

[3] James bach, Exploratory Testing Explained v.1.3 4/16/03 , http://www.satisfice.com/articles/et-article.pdf

[4] State of Testing Survey http://www.practitest.com/pdf/State_of_Testing_Survey_2013_Japanese.pdf

[5] James bach, Session-Based Test Management, http://www.satisfice.com/sbtm/

ソフトウェア開発プロセスの違いによるテストプロセス成熟度の比較・考察

河野　哲也
株式会社　日立製作所
tetsuya.kouno.cb@hitachi.com

高野　愛美
株式会社　日立製作所
manami.takano.pj@hitachi.com

要旨

本稿では，3 つの異なる開発プロセスから少数の QA チームを取り上げ，それらのテストプロセス成熟度の比較・考察を行い，それらの違いを明らかにする．我々の事業部では，多様な顧客要求などに応じて，開発プロセスを柔軟にテーラリングしてきた．しかし，QA 部門では，それら開発プロセスに特化したテストプロセスは定めず，現場の創意工夫によってテストプロセスを適応・改善させてきた．本研究では，それらテストプロセスに対して，テストプロセス改善モデルとしてTPI NEXTⓇを取り上げ，成熟度の評価を行い評価結果の比較・考察を行う．それにより，異なる開発プロセスにおけるテストプロセスの違いを明らかにする．

1. はじめに

様々な製品・サービスを開発するソフトウェア開発では，多様な要求に追従するために，ソフトウェア開発プロセスを柔軟に変更・改善している．また，それに伴い，テストプロセスも変更や改善することが多い．そして，そのような変更・改善を進めるにあたり，プロセス改善モデルを利用して活動を推進することも重要な取り組みの一つであり，またテストプロセス改善技術の関心も高まっている[1].

我々の事業部でも同様に，様々なソフトウェア製品を開発している．一例をあげると，データベースやシステム運用管理ソフトウェアやプラットフォーム管理ソフトウェアなどである．我々が所属する部門は，そのようなソフトウェアを開発する部門とは独立した品質保証部門（以降，品質保証を QA（Quality Assurance）と略す）であり，各製品に対しての QA 業務は，基本的には QA 部門の複数の QA エンジニアで編成された QA チームによって実施されている．

一方で，それら様々なソフトウェア製品を開発するにあたり，多様な顧客要求やビジネス・組織上の制約に合わせて，柔軟にソフトウェア開発プロセス（以降，開発プロセスと略す）をテーラリングしてきた．その結果，我々の事業部では，ウォータフォール（以降，WF を略す）型開発プロセス・反復型開発プロセス・ユーザ機能駆動型開発プロセスの 3 つが代表的な開発プロセスとなっている．これらの開発プロセスの詳細は，2.2 節で述べる．

そのような 3 つの開発プロセスに対して QA 部門では，標準的なテストプロセスはあるものの，それら開発プロセスに特化したテストプロセスは定めず，開発プロセスの変化に従い現場の創意工夫によって QA プロセスを適応・改善させてきた．ここで，そのような適応・改善を進めるにあたり，TPI[2]や TMMi[3]などに代表されるテストプロセス改善のためモデルの導入は行わず，QA チームのリーダの裁量によってテストプロセス改善を進めてきた．

以上を背景として，3 つの開発プロセスに適応させてきたテストプロセスに対して，テストプロセス改善モデルから捉えると特徴的な違いがあるのではないかと考え，本稿ではそれらの違いを明らかにする．具体的には，テストプロセス改善モデルとしてTPI NEXT[4]を採用し，3 つの開発プロセスに対応する QA チームにおけるテストプロセスの成熟度を評価し，評価結果を比較・考察を行う．ここで TPI NEXT を採用するのは，このモデルの特徴としてテストプロセスよりさらに広い視点の評価軸を保有するため，様々な側面で比較・考察ができると考えたからである．

また本稿では，テストプロセスの成熟度の評価結果の比較・考察を行うことをスコープとしている．しかしながら，本研究の大きな目的は，開発プロセスの変化に伴い，テストプロセスを適応・改善してきた QA チームが保有するノウハウや知見，仕掛けなどの実践的工夫を成熟度の評価を行うことにより見える化し，それに基づき他の QA チーム間に横展開することである．加えて，TPI NEXT を利用し，適応・改善の際にテストプロセス上の基本動作の抜け落ちにも気づくこともまた目的の一つである．

つまり，プロセス上のどこにどのような実践的工夫あり，また適応・改善の際にどこのどのような基本動作が抜け落ちたのかを客観的に把握するためのモノサシとして TPI NEXT を利用する．

ここで，図1に研究の全体像を示す．その流れとして，まず 3 つの異なる開発プロセスにおける QA チームを対象に TPI NEXT を利用して評価を実施し，テストプロセス成熟度の評価結果を得る．そして，評価結果データを

整理したのち考察を行い，開発プロセスの違いによるテストプロセス成熟度の差異およびそれに基づく知見を得る．図1は，本稿の章・節の構成に対応させているため，以降，適宜参照されたい．

本稿では，第 2 章で研究対象として，TPI NEXT および 3 つの開発プロセスを概観し，その議論に基づき本研究の目的を示す．第 3 章では TPI NEXT を利用したテストプロセス成熟度の評価の流れを示す．第 4 章では，テストプロセス成熟度の評価結果を第 3 章で述べた研究の目的に基づき比較・考察する．

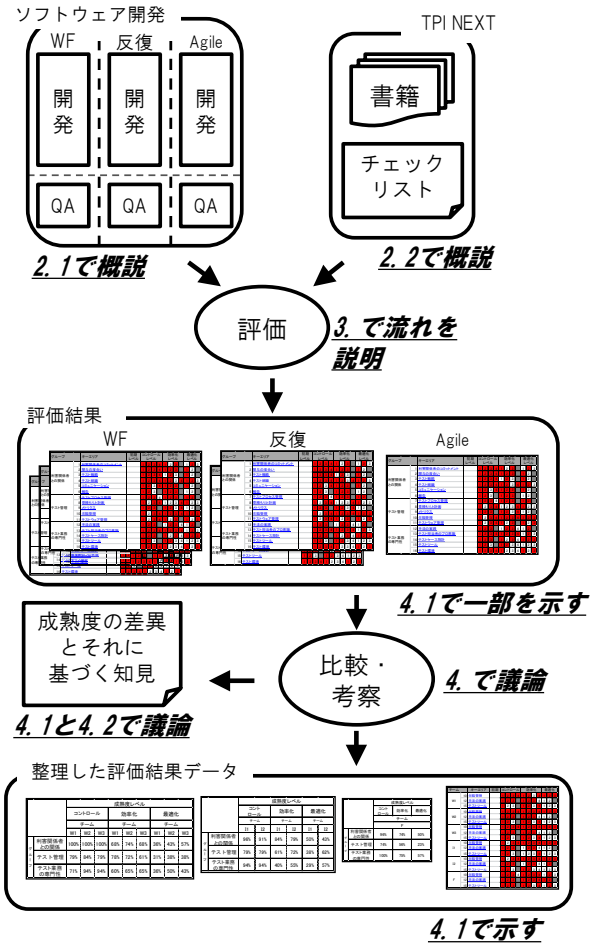


図1　本研究の全体像

2. 研究対象および目的

本章では，テストプロセスの成熟度を評価するためツールと評価する対象の 2 つの視点で研究対象を概観する．すなわち，評価するためツールはTPI NEXT，評価する対象は各開発プロセスにおける QA チームのテストプロセスが対応し，それらについて概説する．そして，その概説に基づき本研究の目的を示す．

2.1. TPI NEXT

TPI NEXT は，BDTPI を基本モデルとして利用することでテストプロセスの成熟度を判断するための標準である．本稿では TPI NEXT において，テストプロセスの成熟度を評価するためのモデルを BDTPI モデルとし，そのモデルを利用するための手順，また評価結果に対する改善提案など手法全体の総称をTPI NEXTとして区別し議論を進める．

グループ		キーエリア	初期レベル	コントロールレベル				効率化レベル				最適化レベル			
利害関係者との関係	1	利害関係者のコミットメント		1	2	3	4	1	2	3		1	2	3	
	2	関与の度合い		1	2	3	4	1	2	3		1	2		
	3	テスト戦略		1	2	3	4	1	2	3		1	2		
	4	テスト組織		1	2	3	4	1	2	3	4	1	2	3	
	5	コミュニケーション		1	2	3	4	1	2	3		1	2		
	6	報告		1	2	3		1	2	3		1	2		
テスト管理	7	テストプロセス管理		1	2	3	4	1	2	3		1	2		
	8	見積もりと計画		1	2	3	4	1	2	3	4	1	2	3	
	9	メトリクス		1	2	3		1	2	3	4	1	2		
	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3	
	11	テストウェア管理		1	2	3	4	1	2	3		1	2	3	
テスト業務の専門性	12	手法の実践		1	2	3		1	2	3	4	1	2		
	13	テスト担当者のプロ意識		1	2	3	4	1	2	3	4	1	2	3	
	14	テストケース設計		1	2	3		1	2	3	4	1	2	3	
	15	テストツール		1	2	3		1	2	3	4	1	2	3	
	16	テスト環境		1	2	3	4	1	2	3	4	1	2	3	

図2　テスト成熟度マトリクス

グループ		キーエリア	初期レベル	コントロールレベル				効率化レベル				最適化レベル			
利害関係者との関係	1	利害関係者のコミットメント		1	2	3	4	1	2	3		1	2	3	
	2	関与の度合い		1	2	3	4	1	2	3		1	2		
	3	テスト戦略		1	2	3	4	1	2	3		1	2		
	4	テスト組織		1	2	3	4	1	2	3	4	1	2	3	
	5	コミュニケーション		1	2	3	4	1	2	3		1	2		
	6	報告		1	2	3		1	2	3		1	2		
テスト管理	7	テストプロセス管理		1	2	3	4	1	2	3		1	2		
	8	見積もりと計画		1	2	3	4	1	2	3	4	1	2	3	
	9	メトリクス		1	2	3		1	2	3	4	1	2		
	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3	
	11	テストウェア管理		1	2	3	4	1	2	3		1	2	3	
テスト業務の専門性	12	手法の実践		1	2	3		1	2	3	4	1	2		
	13	テスト担当者のプロ意識		1	2	3	4	1	2	3	4	1	2	3	
	14	テストケース設計		1	2	3		1	2	3	4	1	2	3	
	15	テストツール		1	2	3		1	2	3	4	1	2	3	
	16	テスト環境		1	2	3	4	1	2	3	4	1	2	3	

図3　評価後のテスト成熟度マトリクスの例

BDTPI モデルでは，テストプロセスの成熟度を評価するための切り口として，16 のキーエリアが用意され，それらは 3 つのグループに分類されている．そして，それらのキーエリアに対して初期レベル・コントロールレベル・効率化レベル・最適化レベルの4つの成熟度のレベルが定義されている．また，各キーエリアの成熟度を判定するた

めに，成熟度レベル毎に 3～4 つのチェックポイントが設けられており，それらのチェックポイントに対して自組織のテストプロセスを照らし合わせ回答し，その成熟度が評価できるような仕組みとなっている．

加えて，各キーエリアに対しての評価結果は，テスト成熟度マトリクスで概観できるようになっている．図2にテスト成熟度マトリクスを示し，評価後のテスト成熟度マトリクスの例を図3に示す．ここで，各キーエリアに対して用意されたチェックポイントでそれを満たすと回答したものが，セルが赤く塗りつぶされているものと対応する．つまり，一つ一つのチェックポイントに回答することで，個別のキーエリアの成熟度，およびグループの成熟度などいくつかの視点で評価結果に対して議論することが可能となる．

一方，段階的改善のための順序の指針として，チェックポイントをグルーピングしたクラスタという考え方を BDTPI で示している．クラスタは，ビジネス上の要因や制約に応じてカスタマイズすることを推奨している．

なお本稿では，評価結果を比較することを目的としているため，改善に関して部分的に議論するものの基本的なスコープには含まれていない．加えて，TPI NEXT は，アセッサなどの専任者による認定は行わず，自己（自組織）評価に基づく自己（自組織）改善を行っていく点が特徴的である[4]．

本研究では，BDTPI で用意されているチェックポイントを抜き出し，整理したチェックリストを用意し，それに回答することで各 QA チームのテストプロセスの成熟度を第 3 章に示す流れに従い評価する．

2.2. 評価対象

本節では，我々の事業部で代表的な 3 つの開発プロセス，WF 型開発プロセス・反復型開発プロセス・ユーザ機能駆動型開発プロセスおよびそれぞれのプロセスにおける QA チームの立ち位置について概説する．

WF 型開発プロセス： 本開発プロセスは，一般的に周知された WF 開発プロセスと大きな違いがないため，詳細は割愛する．QA チームは，開発部門とは独立した立場で QA 業務を遂行している．また，製品の出荷判定は，QA チームの責任者によって実施される．

反復型開発プロセス： 本開発プロセスでは，Unified Process を参照し，ユーザエクスペリエンスの向上・信頼性・保守性の確保に着目し，我々の事業部で考案した開発プロセス[5]に従っている．QA チームは，反復により開発部門と協業する機会が増えるため，WF 型開発プロセスほど独立していないが，基本的な方針として開発部門とは独立した立場で QA 業務を遂行している．また製品の出荷判定は QA チームの責任者によって実施される．

ユーザ機能駆動型開発プロセス： 本開発プロセスでは，Feature Driven Development[6] の考え方と類似する点が多く，ユーザ機能単位に分割されたフィーチャーチームが独立・自律的に開発を進める．各フィーチャーチームは 10 人程度と小規模であり，プロダクトオーナー，開発担当者，QA 担当者といった専門の役割を持ったメンバーで構成され，担当機能のデリバリーに上流から下流まで責任を持つ．スクラムやタイムボックス，エンドゲームなどアジャイル開発のプラクティスを導入している．QA チームは開発部門とは独立しておらず，フィーチャーチームの中の QA 担当者として QA 業務を遂行している．製品の出荷判定は，プロジェクトのメンバーおよびステークホルダーの合議により行われ QA 担当者も発言権を持つ．

以上の 3 つの開発プロセスの特徴を表1に整理する．これらの特徴がそれぞれの QA チームのテストプロセスの成熟度に影響があるのではないかと我々は考えている．

表1　開発プロセスと QA チームの特徴の整理

開発プロセス	アプローチ	QA の立場	出荷判定
WF 型	WF	独立	QA 部門の責任者
反復型	反復	独立	QA 部門の責任者
ユーザ機能駆動型	アジャイル	協業	ステークホルダーによる合議

そして，それぞれの QA チームに共通する標準的なテストプロセスは，テスト計画・テスト設計・テスト実行・テスト報告といった一連の活動で構成される．

本稿では，適用対象として，WF 型・反復型・ユーザ機能駆動型開発プロセスにおける QA チームをそれぞれ 3 チーム，2 チーム，1チームを取り上げ，TPI NEXT に基づきテストプロセスの成熟度を評価し，議論する．ここで議論の簡単のために，WF 型・反復型・ユーザ機能駆動型開発プロセスにおける QA チームをそれぞれ W1・W2・W3 チーム（3 つのチームを総称して W チームと略す）と呼び，I1・I2 チーム（2 つのチームを総称して I チームと略す）と呼び，F チームと呼ぶ．

2.3. 目的

本研究の目的は，3 つの異なる開発プロセスの QA チームにおけるテストプロセス成熟度の比較・考察を行い，その違いを明らかにすることである．この目的のために具体的には，W チーム・I チーム・F チームそれぞれのテストプロセス成熟度の比較・考察を行い，それらの異なる点・

共通する点に関して議論する. また, 開発プロセスに加えて表1が示す QA の立場・出荷判定の違いにも着目する.

以上を踏まえ目的のために次の研究設問を設定する.

RQ1 ： 3 つの異なる開発プロセスで成熟度レベルに顕著な違いがみられるか

RQ2 ： QA の立場, すなわち W チーム・I チームと F チームとで成熟度レベルに顕著な違いがみられるか

RQ3 ： 3 つの異なる開発プロセスで特定のキーエリアで顕著な違いがみられるか

RQ4 ： QA の立場, すなわち W チーム・I チームと F チームとで特定のキーエリアで顕著な違いがみられるか

3. TPI NEXT を用いた評価の流れ

先に述べたように TPI NEXT は, アセッサによる評価を必要とせず, 自組織によってテストプロセスの成熟度を評価するアプローチを採用している. しかしながら, 評価者に対して必要なトレーニングを実施せずに, 評価を進めると「チェックポイントにおける判定の強弱の違い」や「用語の解釈の齟齬」などの課題が考えられる. とはいえ, 本稿では, 複数の QA チームのテストプロセスの成熟度を比較・考察することが目的であるため, QA チーム間でばらつきを減らし共通的な見解を得て評価を進めることが重要である.

以上を踏まえて想定される課題を極力排除するために, 次に示すような流れで TPI NEXT を用いテストプロセスの成熟度を評価することとした. 本評価は, 個別評価と最終評価で構成される.

まず個別評価では各 QA チームのテストプロセス全体を熟知している QA チームのリーダにチェックリストに回答をしながら, テストプロセスをチームリーダ個別で評価する. ここでチェックリストは, BDTPI のモデルの各チェックポイントを抜き出し, 整理したものである.

次に最終評価では, 各リーダは会合形式のミーティングに参加し, 個別評価による各自の評価結果を他のテストプロセスの評価結果と比較・議論し, 各チェックポイントの判断結果とその根拠を確認しながら, 各テストプロセスの最終的に評価をしていく.

4. 比較・考察

第 3 章で示した評価の流れに従い, 6 チームすべてのテストプロセスの成熟度を評価した. 以降, 評価結果の一部もしくは評価結果を整理したデータを示しながら議論を進める. 4.1 節では, 2.3 節で示した目的に基づき比較・考察を行い, 4.2 節では総合的な議論を行う.

4.1. 目的に基づく比較・考察

まず, 全体の評価結果を概観できるようにするために, 次の方法でデータを整理した. 各 QA チームの評価結果に対して, キーエリアのグループと成熟度レベルが交わるチェックポイント群の達成の割合をパーセントで示す. 例えば, 図3において,「グループ:テスト管理」と「レベル:コントロールレベル」が交わるチェックポイント群は 19 ポイントあり, それらの達成は 15 ポイントであるので, そこには 15/19, すなわち 79%という達成の割合を示す. 以上のデータ整理をすべての QA チームのデータに対して実施した. その結果を W チーム・I チーム・F チームごとに表2・3・4に示す.

表2 W チームの適用結果

		成熟度レベル								
		コントロール			効率化			最適化		
		チーム			チーム			チーム		
		W1	W2	W3	W1	W2	W3	W1	W2	W3
グループ	利害関係者との関係	100%	100%	100%	68%	74%	68%	36%	43%	57%
	テスト管理	79%	84%	79%	78%	72%	61%	31%	38%	38%
	テスト業務の専門性	71%	94%	94%	60%	65%	65%	36%	50%	43%

表3 I チームの適用結果

		成熟度レベル					
		コントロール		効率化		最適化	
		チーム		チーム		チーム	
		I1	I2	I1	I2	I1	I2
グループ	利害関係者との関係	96%	91%	84%	79%	50%	43%
	テスト管理	79%	79%	61%	72%	38%	62%
	テスト業務の専門性	94%	94%	40%	55%	29%	57%

表4 F チームの適用結果

		成熟度レベル		
		コントロール	効率化	最適化
		チーム		
		F		
グループ	利害関係者との関係	96%	74%	50%
	テスト管理	74%	56%	23%
	テスト業務の専門性	100%	75%	57%

以降，2.3 節で示した目的に基づき，比較・考察を行う．なお，成熟度レベルが高くなるほど，各チームによるキーエリアの達成度合いのばらつきが大きく，開発プロセスによる共通性が見られないため，比較・考察の際には成熟度レベルの低いところに焦点を当てて議論する．

RQ1 ： 3 つの異なる開発プロセスで成熟度レベルに顕著な違いがみられるか

表2・3・4を比較すると，以下のことが考察できる．

1-1) W チームのすべては，利害関係者との関係のグループでコントロールレベルを達成している．他のチームは達成していない．

1-2）I チームは顕著な違いは見られない．

1-3) F チームは，テスト業務の専門性のグループでコントロールレベルを達成している．他のチームは達成していない．

RQ2 ： QA の立場，すなわち W チーム・I チームと F チームとで成熟度レベルに顕著な違いがみられるか

表2・3・4を比較すると，以下のことが考察できる．

2-1) W チーム・I チームともに F チームと比較すると，総じてテスト管理のグループの成熟度が高く，テスト業務の専門性のグループの成熟度が低い．

RQ3 と RQ4 は，個別のキーエリアの議論になるため，それらの議論に必要なキーエリアの適用結果を抜出し，整理したものを図4に示す．図4では，それぞれの QA チームの結果を比較しやすいように，全ての QA チームにおける 3 つのキーエリア「欠陥管理」「手法の実践」「テストツール」に関する評価結果を抜出し，一覧で表している．

チーム		キーエリア	初期	コントロール				効率化				最適化		
W1	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3
W2	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3
W3	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3
I1	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3
I2	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3
F	10	欠陥管理		1	2	3	4	1	2	3	4	1	2	3
	12	手法の実践		1	2	3		1	2	3	4	1	2	
	15	テストツール		1	2	3		1	2	3	4	1	2	3

図4　評価結果の一部

RQ3 ： 3 つの異なる開発プロセスで特定のキーエリアで顕著な違いがみられるか

図4を概観すると以下のことが考察できる．

3-1) W チームは，欠陥管理のキーエリアでコントロールレベルを達成している．他のチームは達成していない．

3-2) I チームは，欠陥管理のキーエリアでコントロールレベルの 2 つ目のチェックポイントを満たしていない．他のチームは満たしている．

3-3) F チームは手法の実践のキーエリアで最適化レベルにある．他のチームはコントロールを達成しているものの，効率化レベルの 4 つ目のチェックポイントを共通的に満たしていない．

RQ4 ： QA の立場，すなわち W チーム・I チームと F チームとで特定のキーエリアで顕著な違いがみられるか

図4を概観すると以下のことが考察できる

4-1) W チーム・I チームともにテストツールのキーエリアのコントロールレベルの 3 つ目のチェックポイントを満たしていない．しかし，F チームは満たしている．

4-2) W チーム・I チームともに欠陥管理のキーエリアで概ねチェックポイントを満たしている．しかし，F チームは 3 つのチェックポイントを満たしていない．

4.2. 比較・考察に対する議論

本節では，前節の比較・考察の議論を踏まえて，総合的な議論および，実践的工夫や基本動作の抜け落ちという観点で考察を行う．

まず前節では，3 つの異なる開発プロセスの QA チームの TPI NEXT による評価結果に対して RQ1～RQ4 に基づき比較・考察を行った．その結果，ごく限られたサンプル数ではあるがいくつかの違いが明らかになった．その違いに関して，以下で実践的工夫や基本動作の抜け落ちという観点で考察を行う．

1-1)の考察より，WF 型開発モデルから反復型開発モデルおよびユーザ機能駆動型開発モデルへの適応の際に，利害関係者との関係のグループでの基本動作の抜け落ちが考えられる．例えば，I2 チームと F チームでチェックポイント「報告は，書面によって行われている」を満たしていない結果となっている．しかしながら，F チームでは書面による報告の重要性は認識しているものの，業務効率化のために，あえて書面作成を省略し口頭での報告でも問題とならないようにコミュニケーションの工夫を施している．I2 チームにとっては，本評価により，基本動作の抜け落ちに気づくことができた．

1-3)と 2-1)の考察より，W チームと I チームともにテスト業務の専門性の成熟度が低いことが分かった．その背景

として，W チーム・I チームともに開発部門は独立した立場をとっているため，Fチームと比較するとテスト自動化の検討が十分にされているとは言えない．具体的には，F チームではプロジェクト全体でテスト自動化を推進する専任の担当者がアサインされているが，W チーム・I チームではテスト自動化は各チームの裁量で推進している．そのため，W チーム・I チーム共にテスト業務の専門性の成熟度が低い結果となった．これは，4-1)の考察ともつながる．とはいえ，W チーム・I チームともに，テスト自動化に関して課題として認識しており，今後テスト自動化の専任者をアサインするような実践的工夫の推進をしていく予定である．

2-1)の考察より，W チーム・I チームともに開発部門とは独立した立場をとっているため，F チームと比較するとテスト管理面に焦点を当てたテストプロセスになっていることが推察される．これは，4-2)の考察ともつながる．例えば，独立した立場として，欠陥管理のキーエリアに関しては，欠陥データの収集や分析を重点的に行うようなテストプロセスになっている．

3-1)と 3-2)の考察より，反復開発モデルにおける特徴的な基本動作の抜け落ちが考えられる．チェックポイントを見たいしていない背景を QA リーダにヒアリングした結果，WF 型開発モデルから反復型開発モデルへの移行に伴い欠陥を管理するシステムを変更した結果，欠陥に対応するテストケース ID の記述を必須としないようになってしまったようである．よって本評価により，基本動作の抜け落ちに気づくことができた．

以上の考察より，TPI NEXT で各 QA チームのテストプロセスの成熟度を評価することによって，それぞれの QA チームのテストプロセス上の実践的工夫や基本動作の抜け落ちが明らかになることが確認できた．

5. おわりに

本稿では，3 つの異なる開発プロセスから少数の QA チームを取り上げ，それらのテストプロセスの成熟度を評価し，その比較・考察を行った．具体的には，WF 型・反復型・ユーザ機能駆動型の 3 つの開発プロセスから，それぞれ QA チームを 3 チーム・2 チーム・1 チームを取り上げ，それらチームのテストプロセス成熟度を TPI NEXT を用いて評価した．そして，研究の目的に基づき評価結果を比較・考察し，それに対する議論を行った．議論の結果，基本動作の抜け落ちや今後の改善のための指針を得ることができた．

本稿は，異なる開発プロセスにおけるテストプロセスの実践的工夫を横展開するための基礎的な研究であり，今後の発展のための一試行の報告である．

今後の課題としては，大規模なテストプロセス成熟度の評価およびそれに基づく定量的な比較，テストチームにおける体系的な実践的工夫の抽出法の構築およびそれらの横展開の適用，TMMi に代表されるほかのテストプロセス改善モデルとの比較，などが考えられる．

参考文献

[1] 池田ほか，パネルディスカッション：テストプロセス改善技術から探るテストの”改善”とは，ソフトウェアテストシンポジウム 2016 東京, 2016.

[2] Coomen, T. et al., Test Process Improvement －A practical step-by-step guide to structured testing－, Addison-Wesley. (富野(翻訳), テストプロセス改善―CMM 流実務モデル，構造計画研究所)

[3] TMMi Foundation, Test Maturity Model integration (TMMi). (http://www.tmmi.org/pdf/TMMi.Framework.pdf)

[4] Vries, G. D. et al., TPI Next®: Business Driven Test Process Improvement, UTN Publishers. (薮田ほか(翻訳), TPI NEXT® ビジネス主導のテストプロセス改善，株式会社トリフォリオ)

[5] 薮田ほか，チュートリアル：「自己評価」と「自己改善」で始めるテストプロセス改善，ソフトウェアテストシンポジウム 2016 東京, 2016.

[6] 香西ほか，大規模ソフトウェア製品開発向け反復型開発 プロセスと適用 ～高ユーザエクスペリエンスの実現に向けて～, SPI Japan 2011,2011.

[7] Agile Software Development using Feature Driven Development (FDD), http://www.nebulon.com/fdd/index.html

（TPI NEXT は，Sogeti Nederland B. V. の商標です）

大量の状態とイベントを持つプログラムの自動解析とテスト手法の提案 ～想定外のイベントを自動テストする～

下村 翔
デンソー
syo_shimomura@denso.co.jp

古畑 慶次
デンソー技研センター
keiji_kobata@denso.co.jp

松尾谷 徹
デバッグ工学研究所
matsuodani@debugeng.com

要旨

画面遷移を含む *GUI* アプリケーションプログラムのテストを組合せ論理で取り扱うことは，原理的に難しく，順序論理と呼ばれる状態とイベントの時系列を考慮する必要がある．ブラックボックス的な順序論理の組合せ数は膨大になるため，状態やイベントを厳密にモデル化する必要があるが，現場では実践できていない．

本稿では，実際のプログラムから状態とイベントを探索し，モデル化を行う．テストは，得られたモデルから時系列の各ノードに対し，イベントの集合，即ちイベントハンドラが実在するイベントを網羅的にテストする．これにより，ある状態において想定外のイベントに対するテストを自動化できる．さらにテストの効率化のために，*SMT* ソルバの *Z3* を用いて不要なテストケースを削減する．

提案手法を過去に開発したアプリケーションプログラムに適用し，その有効性を確認した．提案手法によりこれまで発見できなかった想定外のイベントによる不具合が明らかになった．

1 はじめに

近年，ソフトウェアは大規模化・複雑化が進んでいる．その中で，品質を確保するためのテスト工程は開発全体の約 4 割程度を占めるという調査結果がある [1][2]．我々は，GUI（Graphical User Interface ）を持つアプリケーションのテストに取り組んでいる．GUI アプリケーションのテストは，そのテスト操作に画面上のマウスの位置やタッチパネルの入力などが存在し，一般的なテストと比べ手間のかかるテストである．

非効率な人で操作に対しては，テスト実行の自動化やテストケースの自動組合せ生成など，様々な自動化手法が提案されている．一方で GUI アプリケーションでは，テスト入力とテスト出力はユーザ操作から生ずるイベントと画面などオブジェクトの状態に左右されるため，組合せによるテストケース生成には限界がある．

本論文の目的は，大量の状態を持ち，かつ，多くのイベント処理を持つプログラムに対して，状態遷移を網羅できるテストケースを自動生成することである．一般的には，状態遷移を正しく表現したモデルを定義し，そのモデルの挙動からテストケースを作成する形式手法が知られている．しかしモデルの作成には専門知識が必要なほか，モデルの作成には追加の工数が必要であるため，現実の開発に適用するのは容易でない．そこで本稿では，仕様ベースのモデル化ではなく，実装ベースのモデル化に基づいたテスト手法を提案する．実装ベースのモデル化とは，実コードに対してマイニング技術を使って，イベント処理や状態を探し出し，その挙動を確認するために必要なテストケースを探索する．提案手法は，実装から状態やイベント処理の抽出処理と，抽出したイベント処理の妥当性を確認する処理で構成する．

本論文の構成は次の通りである．2 章ではテスト対象とする GUI アプリケーションの性質について説明する．3 章は関連研究について述べる．4 章では提案手法のアルゴリズムについて述べ，5 章では提案手法の実装について述べる．6 章で提案手法を実際のアプリケーションに対して適用して手法の有効性を示し，7 章でまとめと今後の課題について述べる．

2 GUI アプリケーションの特徴とテストの課題

GUI アプリケーションは同時に受け付けられる操作が多いため，その操作系列の組み合わせの数は膨大になる．このため手動で GUI アプリケーションのテストケースを作成，実行することは現実的ではない．

ユーザの入力操作をランダムに自動生成したり，あるいは部分的に作成したユーザの入力操作を組み合わせる手法を使うと，テストケース生成を自動化できる．しかし同様に組み合わせの数が膨大になるため，現実的な時間内にテスト実行が終了しない．このため実用上は，自動生成された膨大な数のテストケースに対し，重要なものに絞り込んでテストを実行する必要がある．

実行するテストケースを絞り込む手段として，形式手法に適用できるアプリケーションの動作モデルを用意する方法があるが，現実のアプリケーションにはそのような厳密なモデルは存在しないことが多く，今後作られることも期待できない．

開発者に余計な工数をかけずにテストケースを絞り込める手段がなければ，GUI アプリケーションのテストを効果的に運用していくことは難しい．

3 関連研究

3.1 Capture/Replay

現在の GUI アプリケーションのテスト自動化は Capture/Replay 型のものが主流であり，多くのツールが存在する．Capture/Replay 型のテスト自動化では，まず GUI アプリケーションを実際に操作し，操作内容と期待値を記録してテストケースを作成する (Capture)．その後，記録した操作を再生することでテストを自動実行する (Replay)．しかし Capture/Replay 型のテスト手法は，ユーザ操作の内容やテスト結果の確認方法が GUI のレイアウトに強く依存してしまうことが多く，GUI の変更でテストケースが影響を受けやすいという問題がある．

この点を改善するために，テストケースを作成・管理するための IDE を用意する手法 [3] や，操作対象を実行時に画像認識で探すことで小変更に対応する手法 [4] や，キーワード駆動型テストと呼ばれる手法などがある [5][6]．キーワード駆動型テストでは，テスト実行に必要な情報を外部ファイルに記述する．これらの情報は保守がしやすいよう抽象的に記述され，テスト実行時に自動テストツールが具体的な情報に置き換える．

しかしながら，これらの手法にはテストケース内での条件分岐処理の記述が難しいという問題があり，同じような構造を持ったテストケースが多く生成される傾向がある．特に開発フェーズの初期では GUI が頻繁に変化するため，テストケースの保守にコストがかかるという問題がある．

3.2 モデルベーステスト

Capture/Replay 型のテスト手法の問題を解決するためのアプローチとして，モデルベースのテストケース自動生成手法がある [7]．モデルベースのアプローチでは，開発者がアプリケーションの動作を表すモデルを記述すると，モデルからテストコードを自動生成することができる．

モデルで表現された動作に対して網羅的にテストケースを生成することができるが，モデルの作成には専門知識が必要なほか，モデルの作成には追加で工数がかかる．また，モデルで表現されない動作に対してはテストケースが生成されないため，設計時に想定していなかった入力を受け取ったときに，意図しない振る舞いをしてしまうといった不具合を見つけることはできない．

3.3 ランダムテスト

モデルを使わずにテストを自動化するアプローチとして，ランダムテストやファジングと呼ばれる手法があり [8]，これを GUI アプリケーションに適用する手法も提案されている [9][10]．これらの手法では，アプリケーションへ入力するデータや操作をランダムに生成してテストを自動実行し，異常終了などの想定外の動作が起こらないかどうかを確認する．

テスト入力をランダムに生成することで，テスト実行のための専門知識が必要なくなる他，担当者の先入観にとらわれることなくテストを行うことができるという利点がある．しかし一般に GUI アプリケーションは可能な操作のパターンが多いため，これらの手法を GUI アプリケーションに適用する場合，実行するべきテストケースの数が膨大になり，現実的な時間内にテストが終了しないという問題がある．このため，実用上は生成するテスト入力を適切に絞り込む必要がある．

網羅的にテストケースを生成したあとテストケースを絞り込むのではなく，少数のテストケースから初めて徐々に網羅率を高めていくアプローチとして，遺伝的アルゴリズムを用いたテストケースの生成手法が提案されている．[11]．GUI アプリケーションに入力するイベント列を個体とし，よりテストの網羅率を高める個体を選択していくことで，よりよいテストケースを探すことができる．しかし遺伝的アルゴリズムは問題に適用する一般的な方法がなく，かつ設定すべきパラメータが多いため，テスト対象のアプリケーションごとに最適なパラメータを探さなければならないという課題がある．

4 提案手法

本論文では，設計時に考慮から漏れた異常系の扱いに基づく不具合や，実装時に埋め込まれた不具合を見つけることを目的に，GUI アプリケーションを対象とした自動テスト手法を提案する．提案手法では，GUI アプリケーションの実装から状態遷移モデルを自動生成しつつ，テスト入力となるイベント列を自動生成して自動実行する．到達可能な全ての状態を網羅するまでテストの実行を繰り返し，意図しない不具合動作を発見する．

経験上，状態遷移モデルを手動で作成することはコストが高い．また，アプリケーションの仕様書や設計書は状態遷移モデルを自動で導出できるような形式では書かれていないことが多い．そこで提案手法では，テスト対象のアプリケーションのソースコードのみを入力とする．このアプローチには，テスト実行のために追加で必要な作業が少ない他，新規に作成しなければならないドキュメントがないため，既存の開発プロセスに容易に組み込むことができるという利点がある．

正常系の検査は既に開発プロセスに組み込まれていることが多いため，提案手法は異常系の不具合の検出に重点を置いている．提案手法を既存の開発プロセスに組み込むことで，効率的にアプリケーションの品質を高めることができる．

4.1 提案手法のアルゴリズム

提案手法のアルゴリズムを図 1 に示す．提案手法の基本的なアイデアは，一般に非常に数が多くなるテスト対象のアプリケーションの実装上の状態を適切な基準で同値分割し，得られた同値クラスを状態とする状態遷移モデルを構築することである．アルゴリズムの出力は，アサート文を満たせなかった，すなわち異常動作を引き起こした入力イベント列の集合である．

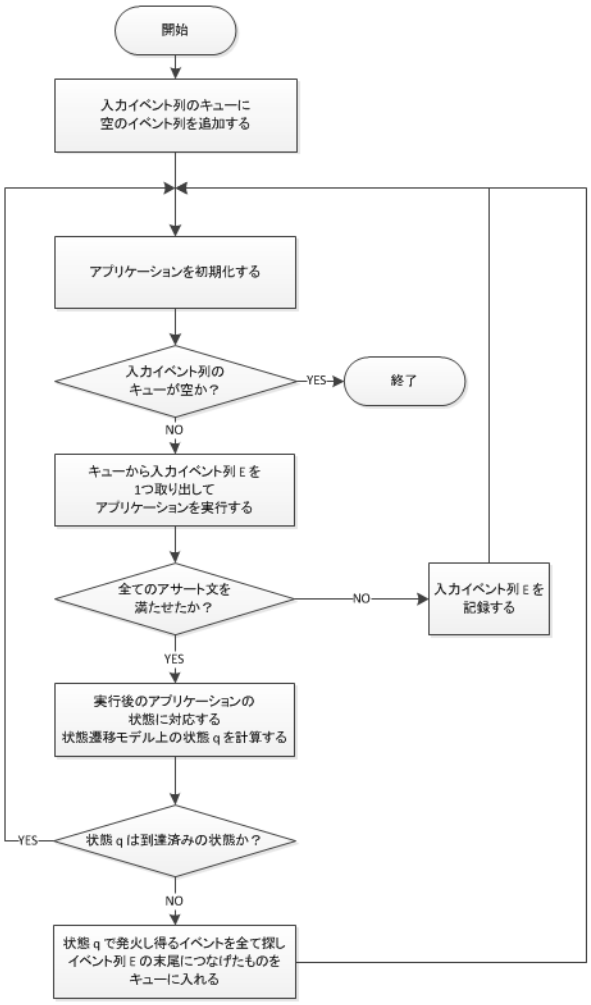


図 1. 状態遷移モデルの自動生成とテストの自動実行のアルゴリズム

キューから取り出したイベント列をテスト入力とし，テスト対象のアプリケーションに対して発火することでテストを実行する．テストの実行中にアプリケーションに埋め込まれたアサート文を満たせなかった場合は，使った入力イベント列を記録する．またアサート文を満たせなかった場合，アプリケーションは異常状態に陥っていると考えられるため，それ以降の探索は行わずに次のテストケースの実行に進む．

全てのアサート文を満たした場合にはアプリケーショ

ンは正常に動作していると判断し，今のアプリケーションの実装上の状態がどの同値クラスに含まれるか，すなわち状態遷移モデル上のどの状態に対応するか計算する．得られた状態遷移モデル上の状態が既に到達したことのある状態であった場合は，それ以降の探索を行わず次のテストケースの実行に進む．新しい状態であった場合は，その状態で発火し得るイベントを全て探索し，それぞれを今の入力イベント列の末尾に追加した上でキューに加える．この処理をキューが空になるまで繰り返すことで，状態遷移モデル上で到達可能な全ての状態に到達できる．

テスト対象のアプリケーションの状態を同値分割する基準は，テスト対象のアプリケーションに合わせて開発者が定義する．例としてウェブブラウザを考えると，例えばアドレスバーに入力された URL の内容で同値分割を行うことが考えられる．この場合，同じ URL のウェブページを開いていれば，例えばページの読み込み状況やスクロール位置などは考慮されず，同一の同値クラスに丸められる．ここで同値分割の基準に URL とページの読み込み状況の組を使うように変更すると，ページの読み込み状況に応じて正しく UI が更新されるか，などのテストも行うことができるようになる．

状態遷移モデルの粒度は同値分割の粒度に依存し，状態遷移モデルを詳細にすると，得られるテストカバレッジは高くなるが，テスト実行時間は長くなる．同値分割の基準を調整しながら上記アルゴリズムの実行を繰り返すことで，テスト時間と得られるテストカバレッジの最適点を探すことができる．この方法には，状態遷移モデルの粒度を変更するためにテスト対象のアプリケーションの実装を変更する必要がないという利点がある．

同値分割の基準が決まれば，状態遷移モデルの構築やテスト実行は全て自動化することができる．実装のポイントは 5 章で述べる．

5　実装

GUI フレームワークである Qt [12] で書かれたアプリケーションを対象とするライブラリとして，提案手法の実装を行った．ただし，提案手法は Qt の機能には依存しておらず，一般の GUI フレームワークに対して実装可能である．

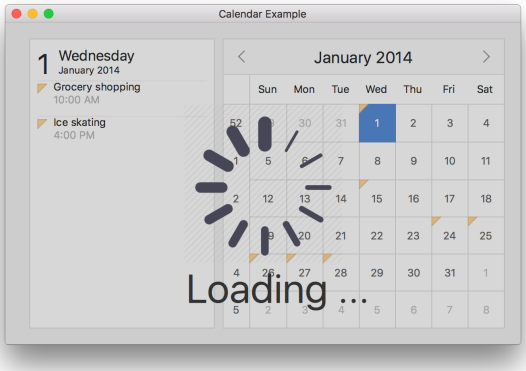


図 2. 上位のコンポーネントが下位のコンポーネントを隠す例

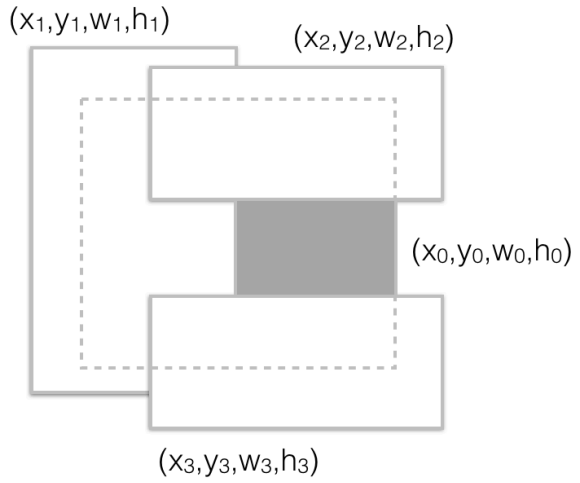
5.1　アルゴリズムの実装

提案手法を実開発に適用するためには，手法の導入・運用コストを低くすることと，テストの実行時間を抑えることが重要である．そのためには，開発者が実装しなくてはならない同値分割処理の実装コストを下げることと，生成する入力イベント列の数を必要最小限に絞り込むことが有効である．

同値分割の基準は，テストの実行時間やテストしたい内容などを考慮して何度も更新されるため，何度も実装を修正する必要がある．この修正コストを下げるために，自動生成された状態遷移モデルを開発者に見やすい形で提示し，開発者がそれを参考にしながら基準を更新できるようにする．

また，発生し得る全てのイベントを取得する処理の実装を工夫することで，状態遷移モデルを網羅できる入力イベント列を生成しつつも，重複していたり実際には起こりえない組み合わせの入力イベント列を取り除く．特に，実際には起こりえない入力イベント列を多く生成してしまうと，テスト実行時間が長くなるだけでなくテスト結果の偽陽性率が高くなってしまう．結果として開発者がテスト結果を無視するようになってしまうため，これを防ぐことは実用上非常に重要である．

以下では，これらの課題を解決するために行った実装上の工夫について述べる．



イベントハンドラを持つコンポーネントの座標が (x_0, y_0, h_0, w_0)，重なった 3 つのコンポーネントの座標が $(x_i, y_i, h_i, w_i)_{[i=1,2,3]}$ で表されるとき，タッチ可能な座標 (x, y) は以下を満たす．

$$\begin{aligned}
&(x_0 < x < x_0 + w_0) \quad \& \quad (y_0 < y < y_0 + h_0) \\
\& \ !((x_1 < x < x_1 + w_1) \quad &\& \quad (y_1 < y < y_1 + h_1)) \\
\& \ !((x_2 < x < x_2 + w_2) \quad &\& \quad (y_2 < y < y_2 + h_2)) \\
\& \ !((x_3 < x < x_3 + w_3) \quad &\& \quad (y_3 < y < y_3 + h_3))
\end{aligned}$$

図 3. GUI コンポーネント同士の重なりと制約式

全イベントの取得処理　テスト対象のアプリケーションがある状態で発火し得る全てのイベントを取得する処理は，その状態で有効な全てのイベントハンドラを取得する処理と同等である．よって，アプリケーション内の全ての GUI コンポーネントを探索して，登録された有効なイベントハンドラを抽出すれば，発火し得る全てのイベントを得ることができる．

しかしながら，GUI コンポーネントを探索して得られた全てのイベントハンドラをそのまま利用すると，得られるイベントの数が非常に多くなってしまう．テストの実行時間を抑えるために，テストに使用するイベント列の数を減らすことが実用上は重要である．

特定のイベントで任意の状態に到達する確率が実数で表されるとき，任意の精度で確率を離散値に丸めることによって，SMT ソルバで効率的に解ける問題に変換する手法が知られている [13]．そこで，この手法を GUI アプリケーションに応用し，探索対象にするイベント数を削減する．具体的には，一般に最も頻繁に生成されるイベントのひとつであるタッチイベントに注目し，GUI コンポーネント同士の重なりを考慮して探索対象にするイベントハンドラを絞り込むよう実装上の工夫を行った．

GUI アプリケーションではユーザの操作を意図的に制限するため，画面の上位に GUI コンポーネントを重ねて下位のコンポーネントを操作させないようにすることがよくある．例えば図 2 のアプリケーションでは，データ読込中はアプリケーション全体に半透明のエフェクトがかかり，アプリケーションを操作できなくなっている．このような場合，下位の GUI コンポーネントに対してタッチイベントが発生することはないため，それらを探索対象から取り除くことができる．

実際にタッチイベントが発生し得る GUI コンポーネントのみを選択するには，その GUI コンポーネント内に他のコンポーネントと重ならない領域が存在するかどうか，という充足可能性問題を解く必要がある．これは例えば，図 3 中の色付きの領域を求めることに相当する．本実装では SMT ソルバの Z3[14] を用いてこの問題を解き，上記問題を満たせない GUI コンポーネントを取り除いた．

タッチイベントのイベントハンドラを探索する処理はライブラリ関数として提供され，開発者が実装する必要はない．また，テスト対象のアプリケーションに固有のイベントを探索対象に加えられるよう，開発者が実装したイベント探索関数を簡単に並用できるように実装を行った．

同値分割の基準　同値分割の基準は開発者が定義，実装する必要がある．基準は通常のプログラムコードとして記述する．具体的には，今のアプリケーションの状態を受け取り，所属する同値クラスを返す関数である．この処理はテスト対象のアプリケーションとリンクし，テスト実行時に使用される．基準の作成には，アプリケーション内の任意の情報を使うことができる．

テストの自動実行　入力イベント列に含まれるイベントを順番に発火させることでテスト対象のアプリケーショ

表 1. 提案手法を組み込んだ開発プロセス

1). アプリケーションの設計
2). アプリケーションのコーディング， アサート文の挿入
3). アプリケーションの実装上の状態を同値分割する基準を定義
4). 状態遷移モデルの自動生成とテストの自動実行
5). 不具合が発見された場合，アプリケーションを修正
6). 得られた状態遷移モデルを元に，必要であれば手順 2 から繰り返し
7). アプリケーション開発の進捗に合わせて手順 2 もしくは手順 3 から繰り返し
8). 機能検査を実施

ンをテスト実行し，アプリケーションの最終状態を得る．この際，イベントハンドラの絞り込みを行った際に得られた実際にタッチ可能な座標の具体的な値をテスト実行時に利用することで，ダミーの座標を使ってタッチイベントを生成する場合に比べてユーザ操作の再現度を高めた．この処理はライブラリ関数として提供されるため開発者が実装する必要はない．

状態遷移モデルの出力　テスト終了後に同値分割の基準を修正するコストを下げるため，アプリケーションの状態とイベントをそれぞれノードとエッジとした状態遷移図として，構築した状態遷移モデルを出力するようにした． 出力形式は HTML で，開発者はブラウザ内でノードを自由に動かして内容を確認することができる．ノードにはアプリケーションのスクリーンショットを使い，生成された状態遷移モデルを開発者が理解しやすくした．

5.2 開発プロセスの実装

提案手法を追加した開発プロセスの例を表 1 に示す．提案手法はコーディングと同時に実行する．提案手法を適用するために追加で必要なプロセスは手順 3 から手順 7 で，追加で作成が必要なドキュメントはない．

手順 4 では，手順 3 で定義される状態遷移モデル上の状態のうち，初期状態から到達可能な全ての状態を探索し，入力イベントをラベルとした状態遷移モデルとして出力する．探索中にアサート文を満たせない入力イベントの列が見つかった場合には，その入力イベントの列も合わせて出力する．

手順 3 から手順 5 まで実行したあとは，必要に応じて状態遷移モデル上の状態の割当てを変更して手順 3 から再実行する．状態遷移モデル上の状態を詳細に分割すれば，より詳細にテストを行うことができるが，それに応じてテストの実行時間は増加する．詳しく確認したい部分に関する状態だけを詳細に分割することで，段階的に状態遷移モデルの粒度を変更することができる．その際，開発者は手順 4 で得られた状態遷移モデルを参照することで，状態分割を効率的に行うことができる．

6 評価

USB メモリ内の写真の一覧表示と拡大表示，スライドショーが行えるイメージビューアアプリケーションに対して提案手法を適用し，評価を行った．このアプリケーションは Qt/QML で実装されており，437 行の C++ コードと 363 行の QML コードからなる．アプリケーションのスクリーンショットを図 4 に示す．

このアプリケーションは，USB メモリが挿入されると USB メモリ内の写真を探索して一覧表示する．一覧表示された写真をタッチするとタッチした写真が拡大表示され，その後任意の場所をタッチすると一覧表示画面に戻る．右下のスライドショー開始ボタンをタッチするとスライドショーが始まり，その後任意の場所をタッチすると一覧表示画面に戻る．

このアプリケーションは，通常のタッチイベントに加え，USB メモリの抜き差しという固有のイベントを持つ．そのため，USB メモリの抜き差しを表すイベントを探索する関数を実装し，探索対象に加えた．この探索関数は 100 行程度で実装することができた．USB メモリの抜き差しを表すイベントが発生した場合の動作は，結合テスト用に実装されていたスタブ実装をそのまま流用した．

また，アプリケーションが満たすべき性質として，以下のようなアサート文を複数埋め込んだ．埋め込むアサート文の量に制限はなく，多ければ多いほど不具合を見つけやすくなる．

- USB メモリ内に画像が 1 枚もないときに，スライドショーボタンが無効になっている
- 拡大表示中もしくはスライドショー表示中に画像が 0 枚になることがない
- 拡大表示中に直接スライドショー表示に遷移しない

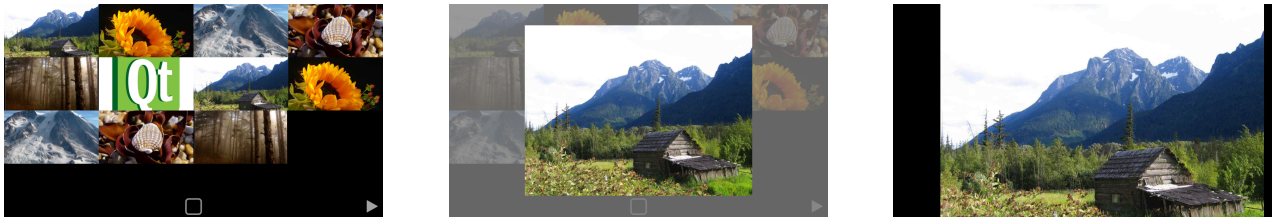


図 4. イメージビューアのスクリーンショット (左から，一覧表示，拡大表示，スライドショー)

評価は 2.4 GHz Intel Core i7 CPU と 16 GB のメモリを搭載したマシンを用いて行った．

試行 1 回目　Qt/QML には，内部状態の変化に応じて GUI を変化させるという動作を実装しやすくする機能がある．この機能を使うと，開発者は内部状態を表す変数の値を変更するだけで，GUI コンポーネントの表示・非表示を切り替えたり，スクリプトやアニメーションが駆動させたりすることができる．テスト対象のアプリケーションもこの機能を使って実装されていた．

そこで，この機能のために実装されていた内部状態を表す変数の値を，同値分割の基準として利用することにした．このアプリケーションは，サムネイル表示中か拡大表示中かを表すブール型の変数と，スライドショー表示中かを表すブール型の変数の 2 つの状態変数を持っていたため，アプリケーションの状態は 4 つに同値分割された．

この同値分割の基準でテストを実行したが，不具合を見つけることはできなかった．得られた状態遷移モデルを図 5 に示す．

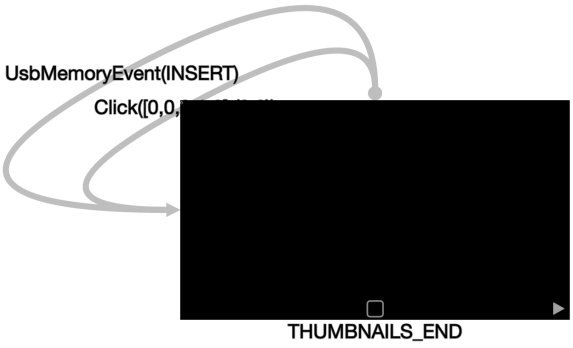


図 5. 1 回目の試行で得られた状態遷移モデル

図 5 において，ノードは各状態におけるアプリケーションのスクリーンショットで，エッジはその状態遷移を起こすイベントの文字列表現で表される．エッジの名称は開発者が必要に応じて自由にカスタマイズできる．また状態遷移モデルはブラウザで表示され，ノードを自由に動かして内容を確認することができる．

図 5 を見ると，スライドショーボタンをクリックしても，USB メモリを挿入しても，同一の初期状態に遷移してしまっており，全てのテストを実行しても初期状態にしか到達できていないことがわかる．これは状態の同値分割の基準が荒すぎたことが原因であり，これでは十分にテストを実行できているとは言えない．

試行 2 回目　1 回目の試行の結果を考慮し，同値分割の基準をより詳細にすることにした．アプリケーションの動作仕様を考慮すると，USB メモリが挿入されているかどうかで GUI が大きく変化するにも関わらず，図 5 では，USB メモリの有無に関係なく同一の状態になってしまっている．

そこで，USB メモリが挿入されているかどうかを同値分割の基準に加えることにした．具体的には，USB メモリが挿入されているかどうかを表すブール型の変数を基準に加え，アプリケーションを 8 個の状態に同値分割した．この同値分割を行う関数は 50 行程度で実装することができた．

この同値分割の基準でテストを実行したところ，拡大表示中またはスライドショー表示中に USB メモリを抜くと，画像が 0 枚になったにも関わらず拡大表示またはスライドショー表示画面にとどまってしまい，画面表示が乱れてしまうという不具合を見つけることができた．得られた状態遷移モデルを図 6 に示す．

このアプリケーションを搭載した環境では，USB メモリが取り外されるとアプリケーション自体が非表示とされていたため，この不具合は顕在化していなかった．提案手法により潜在的な不具合を発見することができた．提案手法は有効であると言える．

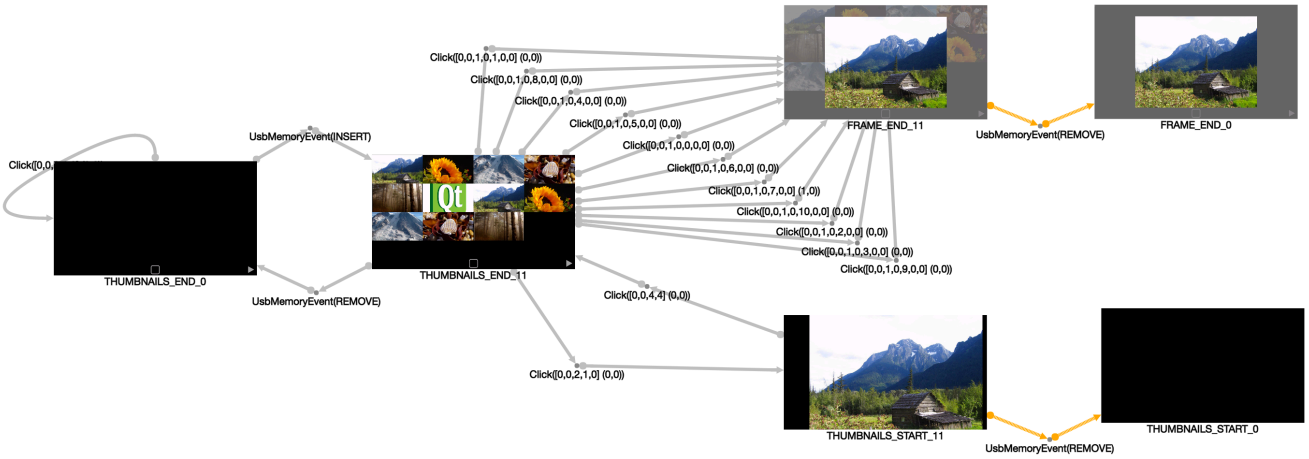


図 6. 2 回目の試行で得られた状態遷移モデル

試行 3 回目　より詳細にテストを実行するため，拡大表示されている写真の名称を同値分割の基準に加えることにした．その結果，不具合に至る操作系列を 23 個見つけることができたが，これらは全て試行 2 回目で見つけられた不具合と意味的に同一のものであった．

試行 2 回目の結果と比較して不具合に至る操作系列の数が増えている原因は，たとえば「1 枚目の写真を拡大表示中に USB メモリを抜く」という操作と「2 枚目の写真を拡大表示中に USB メモリを抜く」という操作が区別されているためである．しかしながら，これらは同じ不具合を複数の操作系列で再現しているに過ぎず，冗長な出力であり，開発者が結果確認にかける工数を増加させてしまう．またテスト自体の実行時間も大幅に増加した．

この結果，どの写真が拡大表示されているかという状態はこのアプリケーションの動作においてはあまり重要ではなく，これを同値分割の基準に加えることは不適切であることがわかった．

性能評価　3 回の試行でのテスト実行結果と，比較のために Z3 によるテストケースの削減を行わなかった場合の試行 2 と試行 3 のテスト実行結果を表 2 に示す．

表 2 において，同一の不具合を起こす操作系列が複数得られた場合は，重複として数えている．また別の GUI コンポーネントの下に隠れた GUI コンポーネントを操作するなど，実際には起こりえない操作を実行した結果，不具合状態に至ったものは偽陽性として数えている．Z3 を使うことで，偽陽性率を低く抑えられることが確認できた．

重複や偽陽性が増加すると開発者が結果確認にかける工数が増加するため，これらを低く抑えつつ，不具合数を増やせる同値分割の基準をうまく探す必要がある．今回の 3 回の試行の中では 2 回目の基準が最適であったが，よりよい基準が存在する可能性がある．

7　まとめと今後の課題

これまで自動化が難しかった GUI アプリケーションのテストに対し，導入コストを抑えて既存の開発プロセスに容易に組み込めるテスト自動化の手法を提案した．経験上，アプリケーションの動作モデルを開発者が手動で作成することはコストが高く，また既存のアプリケーションの仕様はモデルを導出できるほど形式的に書かれることはない．そこで，テスト対象のアプリケーションのソースコードからモデルとして状態遷移モデルを自動で構築し，それを元に網羅的にテストを実行するアプローチを取った．

提案手法の実装にあたり，開発者の負担を抑えることと，テスト結果の偽陽性を抑えることを重視した．実装にコストがかかる入力イベント列の生成部分をライブラリとして提供するほか，GUI コンポーネントの重なりを考慮して余計な入力イベント列を生成しないようにした．

提案手法を過去に開発した試作アプリケーションに対して適用した．アプリケーションは大量の状態を持ち，かつ多くのイベント処理を持つが，現実的な時間で網羅

表 2. 各試行のテスト結果

	到達状態数	実行回数	実行時間	検出された不具合の候補			
				総数	不具合数	重複	偽陽性
試行 1 回目	1	5	4.5 秒	0	0	0	0
試行 2 回目	6	44	36.5 秒	2	2	0	0
試行 2 回目 (Z3 なし)	7	90	84.6 秒	14	2	0	12
試行 3 回目	70	226	527.9 秒	23	2	21	0
試行 3 回目 (Z3 なし)	81	1011	970.2 秒	166	2	21	143

的なテストケースを自動生成することができた．またテスト実行の結果，未知の不具合を見つけることができた．提案手法は有用であると言える．

今後の課題としては，状態の同値分割の基準の決め方と，タイマーイベントやアニメーションの扱い，イベントハンドラ内での分岐の扱いが挙げられる．

状態の同値分割の基準の決め方　同値分割の基準の定義により，テストの網羅度や実行時間は大きく変化する．網羅度とテスト実行時間のトレードオフの中で最適な基準を見つけることが重要であるが，これを機械的に探索する方法は確立できておらず，開発者の経験によるところが大きい．

同値分割の基準を機械的に決める方法を確立することは今後の課題である．

タイマーイベントやアニメーションの扱い　タッチイベントが発生してから時間をおいて UI から応答が発生する場合に，イベントを取りこぼす可能性がある．このような特性を持つイベントハンドラを扱う仕組みを構築することは今後の課題である．

イベントハンドラ内での条件分岐の扱い　現状の実装では，イベントハンドラ内に条件分岐がある場合，全てのパスを網羅的に実行することは保証されない．条件式そのものや，条件式中に現れる変数を同値分割の基準に加えることで，イベントハンドラ内での条件分岐も網羅できる可能性がある．

参考文献

[1] 独立行政法人情報処理推進機構 技術本部ソフトウェア・エンジニアリング・センター監修，＜第 2 版＞ソフトウェア開発データ白書 2010-2011，独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター，2011.

[2] 日本情報システムユーザー協会，ソフトウェア開発管理基準に関する調査報告書（ソフトウェアメトリックス調査），日本情報システムユーザー協会，2011.

[3] “Selenium”. `http://www.seleniumhq.org/`.

[4] 平井潤，関根智，川野晋一郎，“ソフトウェアのテスト効率と精度を向上させる gui 画面の自動操作技術,” 東芝レビュー，vol.63，no.6，pp.36–39，2008.

[5] “RanoRex”. `http://www.ranorex.com/`.

[6] “TestComplete”. `https://smartbear.com/product/testcomplete/`.

[7] E. Bringmann and A. Kramer, “Model-based testing of automotive systems,” Software Testing, Verification, and Validation, 2008 1st International Conference onIEEE, pp.485–493 2008.

[8] K. Claessen and J. Hughes, “Quickcheck: a lightweight tool for random testing of haskell programs,” Acm sigplan notices, vol.46, no.4, pp.53–64, 2011.

[9] E. Alégroth, “Random visual gui testing: Proof of concept.,” SEKE, pp.178–183, 2013.

[10] A. Developers, “Ui/application exerciser monkey,” 2012.

[11] A. Rauf, A. Jaffar, and A.A. Shahid, “Fully automated gui testing and coverage analysis using genetic algorithms,” International Journal of Innovative Computing, Information and Control (IJICIC) Vol, vol.7, pp.●●–●●, 2011.

[12] J. Blanchette and M. Summerfield, C++ GUI programming with Qt 4, Prentice Hall Professional, 2006.

[13] M.N. Rabe, C.M. Wintersteiger, H. Kugler, B. Yordanov, and Y. Hamadi, “Symbolic approximation of the bounded reachability probability in large markov chains,” Quantitative Evaluation of Systems, pp.388–403, Springer, 2014.

[14] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” Tools and Algorithms for the Construction and Analysis of Systems, pp.337–340, Springer, 2008.

Flash メモリ管理における Concolic Testing の活用 ～メモリパターンを含む自動回帰テスト～

西村 隆　　古畑 慶次　　松尾谷 徹
デンソークリエイト　デンソー技研センター　デバッグ工学研究所
nishimura_t@dcinc.co.jp　keiji_kobata@denso.co.jp　matsuodani@debugeng.com

要旨

Flashメモリを含むソフトウェアでは，メモリに書き込まれたパターンの影響を受けることから，網羅的なテストが難しい問題を持っている．ここでは，書き込まれたパターンがプログラムの動作に与える影響を Concolic Testing を用いて解析し，パターンの組合せを特定する．同様に，メモリ管理に与えられた引数（変数）についても Concolic Testing を用いて網羅性の高いテストケース作成することができる．実際のメモリ管理ソフトウェアを疑似したモデルを作成し，メモリパターンを含むテスト入力を自動生成した．このテストの能力を試すためバグを埋め込み，発見できることを確かめたので報告する．

1. はじめに

自動車に搭載されるソフトウェアは，その使われ方から高い信頼性が求められてきた．一方で，自動車に搭載されるソフトウェアは大規模化・複雑化が進んでいる．そのため，高い信頼性を保ちつつ，高い生産性を継続的に実現することが課題になっている[1].

その課題には，テストに関する問題が多く含まれている．テストの問題とは，ソフトウェアの動作に影響を与える因子の数と，因子間の関係によってテスト量が決まる．特に状態変数と呼ばれる順序論理が存在すると，膨大な組合せ数が必要となり，テストの効率を悪化させている．ここで対象とした Flash メモリを含むソフトウェアでは，メモリに書き込まれたパターンが，一種の状態変数として影響を与えることから，網羅的なテストが困難とされていた．

本稿では，Flash メモリに書き込まれたパターンが，どのようにソフトウェアの動作に影響を与えているのかについて，記号実行(Symbolic Execution) の技術を用いて探索し，その影響が生ずるパターンを特定する手法[2]を用いた．具体的には Concolic Testing のツール[3][4]を用いて評価プログラムを作成し，Flash メモリを実メモリで疑似することにより実現した．

実用化においては，テスト結果の期待値の問題，即ち「何が正解なのか？」を解決する必要がある．この問題については，回帰テストによる差分検出による方法[5]を用いて解決した．

2 章では，従来のテストにおける問題点を分析し，3 章で Concolic Testing を用いたメモリ管理ソフトウェア結合テストの方法を提案する．4 章では，単純なメモリ管理制御を行う評価用のサンプルプログラムでの実験と評価結果を示し，最後にまとめと今後の課題について述べる．

2. 現行の結合テスト取り組みと課題

2.1. メモリ管理ソフトウェアにおける結合テスト

メモリ管理ソフトウェアは，図1に示すように関数単体のユニット，複数のユニットの集まりであるコンポーネント，複数のコンポーネントの集まりである結合ソフトウェアで構成されている．テストレベルは，次の 3 段階で実施している．

(1) ユニットテスト
　関数単体に対して， MC/DC カバレッジ 100％の基準でテストケースを設計しテスト実施している．
(2) 単体コンポーネントの結合テスト
　複数のユニットを結合した単体のコンポーネントに対してテスト実施している．関連する外部コンポーネントは，テストドライバやスタブにより実現する．テストケースは，機能仕様と状態遷移に着目し，仕様を網羅する基準でテストケースを設計する．対象の規模が大きく，すぐに組み合わせ爆発を起こすため，テスト技法により合理的にテストケース数を減らす工夫を取り入れている[6].
(3) 複数コンポーネントの結合テスト
　複数のコンポーネントを結合したメモリ管理ソフトウ

ェア全体に対してテスト実施している. テストケース設計基準は(2)と同様である. 本テストは, ECU 実機によるテストと Flash メモリをシミュレートして実施するシミュレーションテストの 2 つの環境で実施している. ハードウェア依存部分の品質確保を ECU 実機によるテストで行い, ロジック部分の詳細をシミュレーションテストにより実施している. シミュレーションテストではテストの自動実行を取り入れ, 合理化を進めているが, 次に示す課題がある.

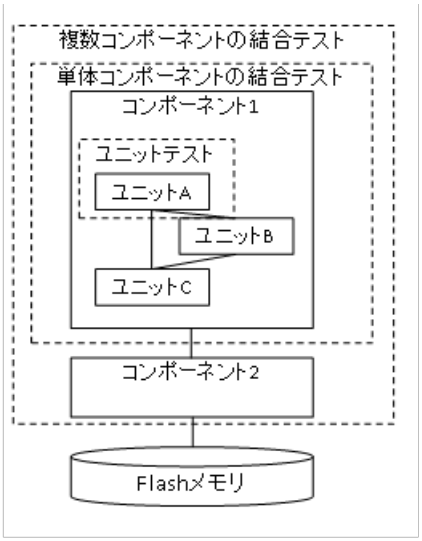


図 1 メモリ管理ソフトウェアのテスト

2.2. メモリ管理ソフトウェアの特徴と課題

状態遷移を含む仕様を基に行うテストケース設計において, 因子間の組合せ基準の問題がある. 組合せを仕様で定義された正しい組合せに限れば組合せ数の爆発は生じない. しかし, 現実問題として, バグは仕様で定義されていない組合せで生じることから, 組合せテストが必要である.

特にメモリ管理ソフトウェアのテスト入力となり得る因子は, プログラム入力であるユーザからの要求(Call 有無, 引数の値), 外部からの応答(応答有無, 応答の値)だけでなく, 図 2 に示すようにプログラム自体が持つ状態である制御の状態変数やタイマ, そして, Flash メモリの格納データもテスト入力となり得る.

メモリ管理ソフトウェアで問題となるのは. Flash メモリの格納データであり, 膨大なメモリ領域を持ち, データのアクセス順序によって取り得る状態は無数となる. 図 3 に示すように, 一旦書き込んだデータに対して上書きができず, 未書き込み領域へ追記する方式を取っているため順序が問題になる.

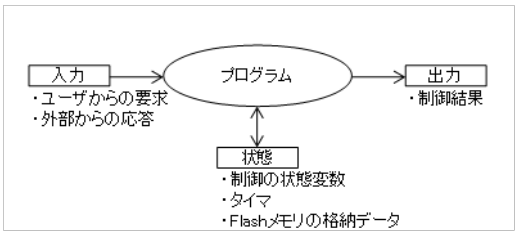


図 2 メモリ管理のテストケースとなり得る因子

ID1	Data	CS	無効
ID1	Data	CS	無効
ID2	Data	CS	有効
ID1	Data	CS	無効
ID1	Data	CS	有効
blank			
blank			
:			
blank			

処理順序
① ID1:書き込み
② ID1:書き込み
③ ID2:書き込み
④ ID1:書き込み
⑤ ID1:書き込み

※最後に書き込まれたデータを有効データとして扱い, それ以前に書き込まれたデータは無効データとする.

図 3 Flash メモリの格納データの例

今までの対策は, ユースケースにより想定しシステムに対して一連のフローを実行したときの様子を記述し, その結果できあがる Flash メモリの格納データパターンや Flash メモリの典型的な故障事象によるエラー推定によりテストケース設計している. 因子の網羅性の面では不十分である課題が残る. 実際の開発においても, Flash メモリの格納データが特殊な配置となってしまった場合において, 設計意図とは異なる動作をしてしまう問題が, 開発終盤の静的検証にて発覚した例もある.

課題としては, 起こり得る組合せを全て網羅する, 合理的で実現可能なテストとすることである.

3. 結合テスト方法の提案

3.1. 実装ベースの回帰テスト

メモリ管理を含む基盤ソフトウェアは，頭出しと呼ばれる新規開発完了後は，実装するマイコン種類の増加に伴う派生開発と，信頼性向上のためのリファクタリングが主な開発となっている．その開発のつど，回帰テストを実施する必要があり，大きな作業割合を占めている．本稿ではこの回帰テストを対象に検討する．回帰テストとは，「変更により，ソフトウェアの未変更部分に欠陥が新たに入り込んだり，発現したりしないことを確認するため，変更実施後，既にテスト済のプログラム対して実施するテスト」である．

回帰テストを実現するためには，網羅性の高いテスト入力値のセットを合理的に生成することが必要である．また，膨大な入力に対するテスト期待値を求めること(オラクル問題と呼ばれる)も必要となる．これらの課題に対するアプローチとして仕様ベースのテストと実装ベースのテストが考えられる．

仕様ベースのテストとは，実装されたプログラムを，仕様を元にテストする手法である．我々が開発するメモリ管理ソフトウェアでは，頭出しの新規開発のときから本手法にてテストケース設計を進め，回帰テストを行ってきた．しかし，本手法には 2 つの課題がある．ひとつは，限られた人にしか対応できない点である．自然言語で書かれた仕様書から，変更の影響がある非互換部分と変更の影響がない非互換部分を正しく抽出するのは困難であり，変更対象プログラムへの深い理解と十分な影響解析が必要なためである．もうひとつは， 2.2 章で述べたように Flash メモリの格納データのような複雑な条件をもつ状態値をテスト入力およびテスト期待値として設定する必要があり，テストケース設計の難易度が高く膨大な工数も必要となる点である．この課題を解決するために，仕様ベースのテストに加えて，実装ベースのテストも実施することを提案する．

実装ベースのテストとは，実装物であるプログラムを静的，あるいは動的に解析し，テストを行う手法である．先行研究の一つに，プログラムが持つ制御構造を探索し，到達可能パスを解析し自動でテスト入力を生成するツールがあり，この有効性が報告されている[7][8][9]．このツールを変更前プログラムと変更後プログラムの双方へ適用し，到達可能パスを網羅的に実行できるテスト入力が自動で生成できる．図 4 に示すように，このテスト入力に対する変更前プログラムのテスト出力と変更後プログラムのテスト出力を比較することで，変更の影響のない互換部分は一致し，変更の影響がある非互換部分は不一致となる．この不一致部分に対して，変更仕様に基づくものか否かの判断を行うことで，回帰テストが可能となる．

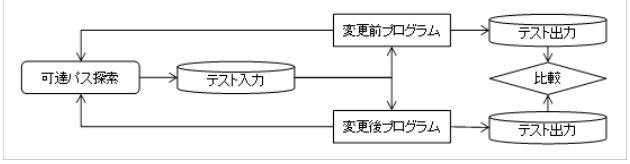


図 4 実装ベースの回帰テスト構成概要

3.2. ツールの選択

到達可能パスを解析し自動でテスト入力を生成するツールの選択を行う．ここでは，現場にて実用化するため，容易に試用できることが求められることから，研修用に提供されている環境の CREST を用いることとした．CREST とは，オープンソースとして公開されている C 言語向け Concolic Testing ツールである．

Concolic Testing は，Concrete Execution と Symbolic Execution を組み合わせた手法である．対象とする変数を指定し，その変数が関与するすべての条件判断に対して，チェックポイントを埋め込み，すべてのチェックポイントを通過する変数値を探索する．チェックポイントを埋め込む前に，ソースコードは抽象構文木に展開している．探索の方法は，制約ソルバを使って，分岐条件を探り，解を求める．制約ソルバで解けない場合は，乱数を使うなどヒューリスティックな手段を併用している．

3.3. メモリ管理ソフトウェアへの適用

Concolic Testing ツールである CREST を用いた実装ベースの回帰テストを，メモリ管理ソフトウェアに適用する方法について説明する．

(1) 対象とする変数の決定

CREST は，テスト入力となり得る変数(以後，CREST 変数と呼ぶ)を指定し，到達可能パスの探索を行う必要がある．メモリ管理ソフトウェアのプログラム入力全てを CREST 変数にできれば，プログラム入力全てを変化させて到達可能パスの探索が可能であるが，実行時間が膨大となり現実的なオーダーではテスト実行できなくなる課題がある． Flash メモリの格納データ以外は，仕様ベースのテストケース設計により，網羅的なテスト入力値を既に有している．

Flash メモリの格納データは，データフォーマットは定義されているものの，膨大なメモリ領域を持ち，データのアクセス順序によって取り得る状態は無数にあるため，人手による仕様ベースのテストケース設計では網羅的な入力値を算出できていない．この課題を解決するために，強力な到達可能パス解析ツールである CREST を活用する．そのため，Flash メモリの格納データを CREST 変数とする．Flash メモリの格納データ以外のプログラム入力は，仕様ベースのテストケース設計で算出した値を予め固定値として与える．

(2) ハードウェア制御のシミュレーション

メモリ管理ソフトウェアは，ハードウェア制御基幹部のソフトウェアコンポーネントであり，ECU 実機において Flash メモリの格納データへアクセスするためには，ハードウェア制御が必要となる．CREST は，C 言語プログラムに対応したツールであるため，そのままでは Flash メモリの格納データを CREST 変数とすることはできない．これを解決するため，ハードウェア制御を実行しているドライバ部をシミュレートしたシミュレーション環境を準備する．シミュレーション環境では，Flash メモリの格納データを RAM 領域として準備し，これを CREST 変数とすることで，CREST の実行が可能となる．

(3) テストドライバの作成

図 5 に示すように，メモリ管理ソフトウェアは，ユーザプログラムからのインターフェース呼び出し（書き込みや読み出し）と周期起動スケジューラからの定期処理（ユーザプログラムからの要求に従い，Flash アクセス制御を順次実行する処理）呼び出しによって実行されるアーキテクチャである．仕様ベースのテストでは，ユースケース想定によりテストケース設計した処理シーケンスをテストドライバで模擬し，テスト実行している．図 6 に仕様ベースのテストドライバ処理具体例を示す．図 6 は，ユーザプログラムから 1 周期目に書き込み処理，2 周期目に読み出し処理したテストケースのテストドライバ処理である．このようなテストドライバは，CREST を用いた実装ベースの回帰テストにおいて 2 つの問題がある．

一つ目の問題は，CREST 変数とした変数に対して，プログラムで代入を行った場合，その時点で CREST による到達可能パスの探索が打ち切られることである．CREST を実行するためには，探索対象プログラムの先頭で CREST 変数を宣言することが必要であるため，図 6 の A 時点で Flash メモリの格納データを CREST 変数として宣言を行う．ところが，1 周期目の書き込み処理で Flash メモリの格納データへ代入してしまうと，B 時点以降の 2 周期目の読み出し処理は，CREST によって与えられた値ではなく，プログラムによって代入された値で動作することになり，CREST による探索ができない．

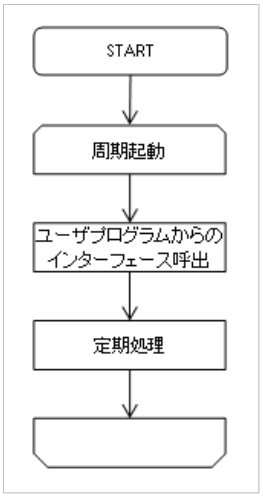


図 5 メモリ管理ソフトウェア動作

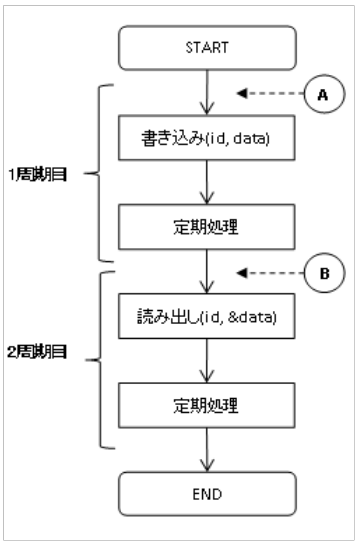


図 6 仕様ベーステストのドライバ処理

二つ目の問題は，メモリ管理ソフトウェアは順序論理を含むプログラムであり，起こり得る組み合わせを全て網羅するためには，爆発的な組み合わせのテストケースを必要とすることである．順序論理を含むプログラムとは，状態を持つプログラムのことである．プログラムの出力は，入力だけでなく以前の実行結果によって決定される状態の影響も受ける．よって，テストケースは，入力パターンに加え処理順序の影響

も受けるため，これを加味したテストケースは組み合わせ爆発を起こす．2.2 章で述べた通り Flash メモリの格納データは，データのアクセス順序によって状態が決定されるため，順序論理を含むプログラムといえる．

これら2つの問題に対して，処理をユーザプログラムの実行単位に分割したテストドライバを用意しCREST 実行することとする．図 6 のような順序制御も，1 周期目，2 周期目それぞれ単体で見ると順序論理を持たない（出力は，入力の組み合わせで一意に決まる＝組み合わせ論理）と言える．これが成立する前提として，分割した処理の途中で外因により状態が変化しないことが必要であるが，メモリ管理ソフトウェアはシングルタスクで駆動されておりこれを満たしている．2 周期目の出力は，1 周期目実行後の Flash メモリの格納データ（状態）によって決まるが，1 周期目と切り離して考えると，Flash メモリの格納データ（状態）も2周期目の入力の一つと考えることができる．よって，処理をユーザプログラムの実行単位に分割したテストドライバであれば，組み合わせ論理のプログラムとなるため，2 つの問題は解消される．一つ目の CREST 探索打ち切り問題は，組み合わせ論理であれば，解消される．プログラム中に代入があったとしても，シングルタスクで駆動されている中では，プログラム中の代入文以降は固定ロジックでありテスト不要なものといえる．二つ目の組み合わせ爆発も，順序論理から組み合わせ論理に置き換えることで解消される．組み合わせ論理となることで，処理順序の組み合わせが不要となり，組み合わせ爆発は起こらない．変更前プログラムと変更後プログラムにおいて，処理の実行単位毎に，全ての分岐条件を網羅した入力に対する出力が一致していれば同じプログラムと言えるため，結合されたソフトウェアに対する回帰テストができているといえる．

テストドライバ作成についてまとめると，図 7 のようになる．最初にテスト入力値設定処理を行う．テスト入力値としては，予め固定値として与えるものとCERST 変数宣言するものがある．次に，テスト対象プログラムを実行し，最後にテスト出力を行う．

4. 提案手法の実験

先に示した実装ベースの回帰テストについて実験し評価を行う．実験には，分析や評価の容易性のため，メモリ管理ソフトウェアを疑似した評価用のサンプルプログラムを準備し，これを使用した．サンプルプログラムは，3 つのコンポーネントで構成し，全体の規模は 615 ステップである．実行環境は，Windows7 上で Vagrant を用いて構築した Ubuntu15.04 の LINUX 環境とした．

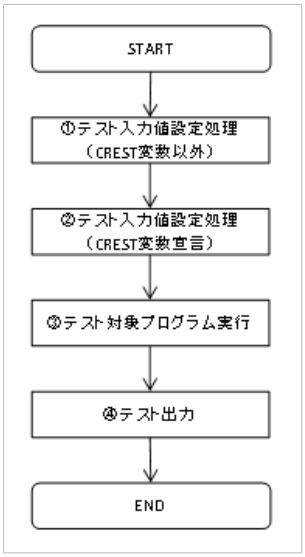


図 7 CREST テストドライバ概要

4.1. テストドライバの実装

3.3 章の方針に従い CREST のためのテストドライバを準備する．テストドライバは簡単で，図 8 のようである．2 行目は，仕様ベースのテストケース設計により，あらかじめ準備した Flash メモリの格納データ以外のテスト入力値設定処理（図 7-①に対応）である．4～15 行目は，Flash メモリの格納データを CREST 変数として宣言している（図 7-②に対応）．17～18 行目は，処理の実行単位に分割したテスト対象プログラムの実行（図 7-③に対応）である．20 行目は，テスト出力処理（図 7-④に対応）である．

```
1       /* set inputdata */
2       setTestData(i);
3       /* declare CREST value */
4       CREST_unsigned_char(flashdata[0]);
5       CREST_unsigned_char(flashdata[1]);
6       CREST_unsigned_char(flashdata[2]);
7       CREST_unsigned_char(flashdata[3]);
8       CREST_unsigned_char(flashdata[4]);
9       CREST_unsigned_char(flashdata[5]);
10      CREST_unsigned_char(flashdata[6]);
11      CREST_unsigned_char(flashdata[7]);
12      CREST_unsigned_char(flashdata[8]);
13      CREST_unsigned_char(flashdata[9]);
14      CREST_unsigned_char(flashdata[10]);
15      CREST_unsigned_char(flashdata[11]);
16      /* execute testcase */
17      ret = ms_write(id, data);
18      ms_periodic();
19      /* output */
20      printf("i = %d, ret = %d¥n", i, ret);
```

図 8　CREST のためのテストドライバ

4～15 行目の CREST 変数の宣言に工夫がある. Flash メモリの格納データは, シミュレーション環境において連続の RAM 領域に確保しているため, CREST 変数としては, unsigned char 型の変数として 1byte ずつ宣言している. Flash メモリは, 通常数十 Kbyte 程度と膨大であり, 全ての領域を CREST 変数として宣言し探索を行うことは現実的な実行時間ではできない. ユーザが取り扱う Flash メモリのデータは, 固定サイズで, Flash メモリ内のデータフォーマットも固定フォーマットであり, これを保持データ数分繰り返している（図 3 のイメージ）. この Flash メモリのデータフォーマット特徴に着目し, 同値分割の考え方より, 実際の Flash メモリが保持できるデータ数よりも小さいサイズを設定することとした. これにより, CREST 変数の個数を削減し, 実行時間の短縮を図っている. このとき, Flash メモリの格納データを何データ分確保すべきか（どこまでデータ数を削減できるか）を決定する必要がある. メモリ管理ソフトウェアには, 複数データを保持している場合のみ動作するロジックが実装されている. 実装ベースのテスト入力としては, 実装されたプログラムの分岐条件を網羅するデータが必要である. これの分析にも CREST が活用できる. Flash メモリの格納データ数を 1 個ずつ増やして, CREST による探索を行う. このときの分岐網羅率を計測し, この値が頭打ちとなったところで, データ数に依存するロジックが十分に実行できるデータ数であると判断し, テスト入力として採用する. 表 1 にサンプルプログラムの Flash メモリデータ数と分岐網羅率の計測結果を示す. 分岐網羅率が頭打ちとなったデータ数は 2 個であり, サンプルプログラムが持つ, 複数データを保持している場合のみ動作するロジック「書き込み済データの CheckSum が異常であった場合, 他の書き込み済データがないかを探索する処理」を持つ設計と一致しており妥当なデータサイズと言える. この考え方を抽象化すると, Flash メモリの格納データは, 一定の形式で定義されたデータの集合体といえる. このようなデータの集合体を入力とし, 分岐条件へ影響を与える実装を持つプログラムへ, 本手法は適用可能である. 具体的には, データベース管理や文字列探索プログラムへ応用できる.

表 1　Flash メモリデータ数と分岐網羅率

データ数	1 個	2 個	3 個	4 個
Flash メモリサイズ	4byte	8byte	12byte	16byte
Iteration 数	56 回	2112 回	57824 回	60000 回以上
実行時間	1.6s	12.2s	320.3s	打ち切り
分岐網羅率	93.55%	100.00%	100.00%	100.00%

4.2. 評価

CREST の実行結果を評価する.

これまでの仕様ベースのテストで抽出したテスト入力（Flash メモリ格納データは固定値採用）時の分岐網羅率と CREST により生成されたテスト入力（Flash メモリの格納データの到達可能パスを CREST により探索した入力値採用）時の分岐網羅率を表 2 に示す. 両者を比較したところ, CREST 変数採用時の分岐網羅率は 100%であり固定値採用時よりも分岐網羅率が高い. この結果より, Flash メモリの格納データのような複雑な条件をもつ状態値を人手によりテスト入力として設定するのが困難な場

表 2　CREST による分岐網羅率

Flash メモリの格納データ	固定値採用	CREST 変数採用
実行時間	1.2s	13.2s
分岐網羅率	70.19%	100.00%

合でも，到達可能パスを解析し自動でテスト入力を生成する CREST のようなツールを利用することで，分岐網羅率が高いテスト入力値を生成できることが確認できた．

さらに，実際の基盤ソフトウェア開発で，開発中に発生した「変更によるインターフェース不一致」が実装ベースの回帰テストにより発見可能かを検証した．今回のサンプルプログラムにおいて，下位コンポーネントに機能追加し，処理成功時の戻り値のパターンを増やす変更を行った．上位コンポーネントは，下位コンポーネントからの戻り値のパターンが増えているため，これに対応する必要があるが，この変更が漏れた状況で，実装ベースの回帰テストを行った．その結果，表 3 に示す変更前プログラムと変更後プログラムのテスト出力にて，互換部分であるはずのテストケース 5 において不一致を検出し，今回の変更漏れが検出できることが確認できた．

表 3 変更前と変更後プログラムの回帰テスト結果

テスト番号	変更前 テスト出力	変更後 テスト出力	比較結果
1	0	0	一致
2	0	0	一致
3	0	0	一致
4	1	1	一致
5	1	2	不一致
6	0	0	一致
7	0	0	一致
8	0	0	一致
9	0	0	一致
10	0	0	一致
11	1	1	一致
12	0	0	一致
13	0	0	一致
14	0	0	一致
15	0	0	一致
16	1	1	一致

5. おわりに

C 言語向け Concolic Testing ツールである CREST をメモリ管理ソフトウェアに適用し，実装ベースの回帰テストが実行可能であることを，サンプルプログラムを使って確認することができた．これにより，Flash メモリの格納データのように，非常に多数のパターンを取る状態変数を持つプログラムにおいて，分岐網羅率を向上したテスト入力が自動で生成でき，ユニットやコンポーネントを結合した状態での，結合時のインターフェース不一致や相互作用により発生する欠陥を摘出するためのテストが可能であることを確認できた．

今後，本提案を実際の開発で利用可能とするために，Flash メモリの格納データサイズや探索対象プログラムの規模増加時の実行可能な上限の調査，および，テストドライバ作成・回帰テスト実行の効率化を検討する予定である．

参考文献

[1] 菅沼賢治，村山浩之，“自動車組み込みソフトウェアにおける開発戦略”，情報処理，vol.44，no.4，pp.358-368，2003.

[2] C. Cadar, P. Godefroid, S. Khurshid, C.S. Pasareanu,K. Sen, N. Tillmann, and W. Visser, “Symbolic execution for software testing in practice: preliminary assessment”, Proceedings of the 33rd International Conference on Software EngineeringACM, pp.1066-1071 2011.

[3] K. Sen and G. Agha, “Cute and jcute: Concolic unittesting and explicit path model-checking tools”, Computer Aided VerificationSpringer, pp.419-423 2006.

[4] CREST, “Concolic test generation tool for c”, http://jburnim.github.io/crest/.

[5] 松尾谷徹，増田聡，湯本剛，植月啓次，津田和彦，“Concolictesting を活用した実装ベースの回帰テスト:～人手によるテストケース設計の全廃～”， ソフトウェア・シンポジウム 2015in 和歌山 SEA:ソフトウェア技術者協会，pp.47-56,2015.

[6] 中野 隆司・小笠原 秀人・松本 智司，“組み合わせテスト技術の導入・定着への取り組み，および上流設計への適用検討の事例”，SQiP シンポジウム 2011，2011

[7] 丹野 治門・星野 隆・Koushik Sen・高橋 健司，「Concolic Testing を用いた結合テスト向けテストデータ生成手法の提案」，Vol.2013-SE-182 No.6，2013

[8] 岸本 渉，「安全系組み込みソフトウェア開発におけるユニットテストの効率化」，ソフトウェア・シンポジウム 2015，2015

[9] 榎本 秀美，「記号実行を用いたテストデータ自動生成の試行評価」，ソフトウェア品質シンポジウム 2015，2015

保証ケース作成支援方式について

山本 修一郎
名古屋大学大学院情報科学研究科
syamamoto@acm.org

要旨

本稿では，独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）の支援を受けて実施した「保証ケース作成支援方式の研究」の主な成果について述べる.

1. はじめに

保証ケース作成支援方式の研究の位置付けを図 1 に示す。保証ケースの導入では、①保証ケースを導入しようとする組織とその能力、②保証ケースによって保証しようとする対象システムの状況、③保証ケースの状況について明確化する必要がある。

十分な能力がない組織に保証ケースを導入することはできない。対象システムの状況では、対象システムのモデルがある場合とない場合がある。モデルがある場合には、モデルに対して保証ケースを作成できる。モデルがない場合には、対象システムのコードに基づいて保証ケースを作成する必要がある。保証ケースがすでに作成されている場合と、新規に作成する場合がある。

以上から本研究では，モデルに基づく保証ケースの統一的作成法，コンポーネントコードに対する保証ケース作成法，保証ケースのレビュ手法，これらの手法に対する教材，保証ケースの導入準備能力評価指標を具体化した（図 2）. なお、保証ケースについては、文献[11,12,13]を参照されたい。

2. 保証ケース作成支援方式の研究内容

2.1. モデルに基づく保証ケースの統一的作成法[1-3]

モデル図の構造情報に基づいて保証ケースの作成活動プロセスを定式化（図 3）し，支援ツールの試作（図 4）により，自動化範囲と自動化による改善効果（図 5）を明確化した.

2.2. コードに基づく保証ケース作成法[5]

コードに基づく保証ケースの作成法を定式化した（図 6）.

2.3. 保証ケースの客観的なレビュ手法[6,7]

保証ケースの構成情報に基づいて，保証ケースのレビュプロセス（図 7）と評価指標（図 8）を定式化した.

2.4. 開発技術者向け教育研修教材の試作[8]

定式化した保証ケース作成・レビュ手法に基づく研修教材 2 件（図 9,10）を作成し，研修実施により有効性を確認した. なお、コードに基づくレビュ手法については、研究資源の制約から省略した。

2.5. 保証ケース導入準備能力評価指標[10]

保証ケース導入準備能力評価指標（図 11）を作成し，組織担当者へのヒヤリングにより，保証ケースの導入可能性を評価した（図 12）.

3. おわりに

本研究は 2015 年 6 月に着手し，2016 年 1 月に完了している. 今後，本研究の成果を順次公開していく予定である.

参考文献

[1] Shuichiro Yamamoto, Assuring Security through Attribute GSN, ICITCS 2015, pp.1-5, 2015
[2] Shuichiro Yamamoto, An approach to assure Dependability through ArchiMate, Assure 2015, pp. 50-61
[3] 山本修一郎,森崎修司,渥美紀寿,正田稔,モデルに基づく統一的保証ケース作成手法の提案, AI 学会, KSN 研究会, 2015. 10
[4] 山本 修一郎,森崎修司, 渥美紀寿, 構成情報に基づく保証ケースレビュ手法の提案,AI 学会, KSN 研究会, 2015. 10
[5] 宮林 凌太, 渥美 紀寿, 森崎 修司, 山本 修一郎, 入力分析に基づくコード保証方法の提案, KBSE 研究会, Vol.115, No.281, KBSE2015-39, pp.17-22, 2015
[6] 山本修一郎, 要求リスクコミュニケーション, KBSE 研究会, KBSE2015-37, pp.7-12 , 2015
[7] Shuichiro Yamamoto, An assurance case review method using Systemigram, AAA2015
[8] 山本 修一郎, 森崎 修司, 渥美 紀寿,保証ケースの先端研修教材の試作と評価, AI 学会 KSN 研究会,2016,3.1
[9] 山本修一郎, 森崎修司, 渥美紀寿, 近藤純平, 大林英晶, GSN レビュ実験と評価について, AI 学会 KSN 研究会, 2016, 3.1
[10] 山本 修一郎, 保証ケース導入準備能力評価指標の提案, 信学技報, KBSE2015-63, 2016-03-04, pp.85-90
[11] 山本修一郎, システムとソフトウェアの保証ケースの動向, [Kindle 版], Amazon Services International, Inc., 2013
[12] 山本修一郎, 保証ケース作成上の落とし穴, [Kindle 版], Amazon Services International, Inc. , 2013
[13] 山本修一郎, 保証ケース議論分解パターン, [Kindle版], Amazon Services International, Inc. , 2013

VDM++仕様から C#コードを生成するツールの開発と評価

千坂 優佑　　大友 楓雅　　力武 克彰　　岡本 圭史
仙台高等専門学校
{a1502020, a1602006}@sendai-nct.jp　　{yoshiaki, okamoto}@sendai-nct.ac.jp

要旨

本報告では，VDM++仕様から C#コードを生成するツール（以下，本ツール）の開発と妥当性・有用性評価について報告する．本ツールを用いることで，VDM++仕様から C#コードを生成でき，CodeContracts による事前・事後・不変条件を用いた生成コードの検証が可能となる．

1. はじめに

情報システムの信頼性向上の有効な手段として，形式手法がある．形式手法の 1 つである形式仕様記述は，仕様を VDM++等の形式仕様記述言語で記述することで，仕様の厳密化や計算機による検証を可能とする．

VDM++を扱う既存のツールには， VDM++で記述された仕様から Java または C++のコードを生成する機能がある．この機能により，仕様を基にコードを作成する工数を削減するだけでなく，その過程でのヒューマンエラーの混入を予防できる．しかし，現状のコード生成機能では Java と C++しか出力できず，また操作定義の事後条件は変換できないなどの制約もある．

本報告では，VDM++仕様から C#コードを生成するツールの開発について報告する．本ツールが VDM++仕様における事前・事後・不変条件を生成コード中の契約へ変換し，C#から利用可能な CodeContracts を用いることで，検証に事前・事後・不変条件を利用できる．

2. コード生成の方法

コード生成処理は，構文解析，抽象構文木の変換，C#コード生成の 3 つの手順で行われる．構文解析および C#コード生成の実装には，既存のツール VDMJ および.NET コンパイラープラットフォーム Roslyn を利用する．

VDM++仕様は 1 つ以上のクラスから成り，クラスはいくつかの定義ブロックから成る．それらの要素ごとに対応する C#の要素を定め，変換を行う．VDM++の事前・事後・不変条件は，CodeContracts においてそれらを表す Contract.Requires・Contract.Ensures・Contract.Invariant に変換する．VDM++の事後条件において返り値を表す RESULT は，メソッドの返り値を表す Contract.Result に変換する．また，操作定義の事後条件において変数の旧値にアクセスするために用いられる旧名称は，Contract.OldValue に変換する．

表 1. コード生成結果の例（一部）

post Count = Count~ + 1;
Contract.Ensures(Count==Contract.OldValue(Count)+1);

3. 本ツールの評価

本ツールによるコード生成の妥当性確認のため，VDM++仕様と本ツールにより生成された C#コードの等価性を調査した．具体的には，簡単な VDM++仕様に対して仕様アニメーションによるテストを行い，本ツールが生成した C#コードに対しても同じ事前・事後条件を用いて単体テストを行ったところ，それぞれ同一の入力に対して同一の出力が得られ，同一のテスト結果が得られた．

次に本ツールの有用性を評価するため，3 つのクラスから成るシステムを対象として従来開発(A)および本ツールを用いた開発(B)を実施し比較した．(A)では，作成済みの UML モデルに基づき 18 行の C#スケルトンコードを作成するのに 15 分程度の時間を費やしたが，(B)では 1 分もかからずに生成できた．さらに，(A)では UML に事前条件が記述されていなかったため事前条件を満たさないような入力も含めたテストケースが作成されたが，(B)では CodeContracts により事前条件が保持されていたためそのようなテストケースは作成されなかった．

4. まとめ

本報告では，VDM++仕様から C#コードを生成するツールを開発し，その妥当性・有用性評価を行った．本ツールを用いることで，開発期間が削減された．また，優先度の高いテストケースに集中できることが簡単な例題で確認できた．

システム理論的因果律モデル STAMP に基づくプロセス改善分析

日下部　茂
長崎県立大学
kusakabe@sun.ac.jp

荒木　啓二郎
九州大学
araki@ait.kyushu-u.ac.jp

要旨

プロセス改善のための分析に，システム理論的因果律モデル *STAMP* を用いるアプローチを提案する．継続的にプロセスを改善し最適化できる状態が目標であることを前提に，プロセスの状態を表現するモデルを用いて目標としている状態に向けた制御ができるかを分析する．提案手法の有効性については，これまでのプロセス改善のアプローチと同様のことが，提案するアプローチで系統的に導くことができるかという観点で評価する．

1. はじめに

ソフトウェアの開発効率の向上や品質の向上には要素技術だけでなく，開発プロセスも含めたライフサイクルプロセスが重要とされている．プロセスは，一旦構築すればそれで終わりということはなく，状況に応じてテーラリングや改善を継続的に行うことが重要である．本研究では，プロセスの状態を表現するモデルを用い目標としている状態に向けた制御ができているかを，システム理論的因果律モデルで分析するアプローチを提案する．

プロセス改善の一つのアプローチとして、最初は標準的なプロセスのプロセステンプレートを採用し，その定着後はそれをベースに継続的に改善を続けるものが考えられる．その際，プロセスデータを測定しながら改善を行う仕組みが埋め込まれたテンプレートを使っていれば，新しい技術の導入の得失をデータに基づいて判断しやすく，適切な導入が実現しやすいと考えられる．そのようなプロセスのテンプレート例として，個人レベルでは PSP(Personal Software Process)[1]，チームレベルでは TSP (Team Software Process)[2] といったものがある．このような考えのもと，我々は形式手法のような新しい技術の導入によるプロセスの改善についての取り組み行ってきた [3] [4] .

しかしながら，個人や組織の開発プロセスに，形式手法のような新しい技術や手法を取り入れ定着させるには，関係者を適切に動機づけることも重要とされている．このような観点から，動機づけ理論のうち，過程理論の一つを基礎として，動機づけプロセスの状態遷移モデルを提案し，新しい技術や手法の導入から定着に至るプロセスの分析に用いた研究もなされている [5].

本研究では，前述の関連研究のように状態を用いたモデルを用いるが期待と，実際の活動を行う側の実際には差があるという観点でモデルを考える (図 1 参照)．常に期待と実際の間にはギャップがあり，順調なのは適宜フィードバックを得ながら監視と制御を行いギャップを許容範囲に収めているからと考える．このギャップが大きくなり過ぎた結果，受容できない損失が生じることをアクシデントとしてモデル化する．開発プロセスに関して期待する状態と，実際の状態のギャップが大きく，アクシデントが起こりかねない状態をプロセス改善のハザードと考え，そのハザードを防ぐことに着目する。

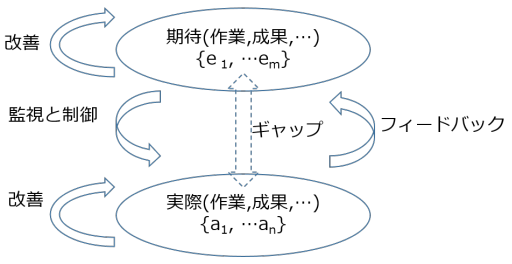


図 1. 期待と実際のギャップに対する監視と制御

前述の観点に基づきシステム理論的因果律モデル STAMP(Systems-Theoretic Accident Model and Pro-

cesses) [6] を用いてプロセス改善を分析するアプローチを提案する．STAMP を利用した分析を行うことで，体系的にプロセス上での脆弱な箇所をハザード要因として識別し，改善すべき点を明らかにすることができると考える．提案手法の有効性については，これまでのプロセス改善の事例と同様の改善が，提案の分析で系統的に導くことができるかという観点で評価する．

本稿の構成は以下の通りである．第 2 節で，システム理論的因果律モデル STAMP について説明する．第 3 節で，個人レベルのプロセス改善フレームワーク PSP を紹介し，第 4 節で PSP に STAMP にもとづいた分析を適用する例について論じる．第 5 節でまとめを行う．

2. システム理論的因果律モデル STAMP

2.1. STAMP 概要

STAMP は MIT の Nancy Leveson 教授によって提唱されたもので，従来の解析的還元論や信頼性理論ではなくシステム理論に基づき，システムを構成するコンポーネント間の相互作用に着目した事故モデルである．STAMP のモデルは，安全制約，階層的なコントロールストラクチャ，プロセスモデルという三つの基本要素で構成されている．

- コントロールストラクチャ：システムの構成要素間の構造と，相互作用を表したもの
- プロセスモデル：コントロールする側がその対象プロセスをコントロールするアルゴリズムと対象プロセスを (抽象化して) 表現したもの
- 安全制約：安全のために守るべき制約

STAMP では，コントロールストラクチャとプロセスモデルに対して，システムの安全制約が正しく適用されているかどうかに着目する．STAMP のコントロールストラクチャとプロセスモデルの基本形を図 2 に示す．

コントロールストラクチャは，一般には上述の基本形を組み合わせ，システム内の複数階層の複数コンポーネントの構造とやり取りされる制御の指示やフィードバックなどの関係を表す．STAMP では，安全に関するコントロールストラクチャがシステムの安全制約を守れず，システムがハザード状態になることでアクシデントが発生すると考える．

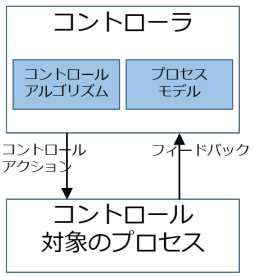


図 2. コントロールストラクチャ

STAMP では，システム内だけでなく，開発や運用のプロセスにおけるコントロールストラクチャも含めて統一的にモデル化できる．そのため，エンジニアリングの観点での要因分析や，人や組織間の関係，規則までを含めた要因分析などが可能である．図 3 にシステムの開発と運用におけるコントロールストラクチャの例を示す．

STAMP 自身はアクシデントを説明するモデルであるが，STAMP をベースとした解析の道具立てやプロセスは複数提案されている (図 4 参照)．STAMP を用いた安全制約の分析は，システムだけでなく，その開発プロセスや運用プロセスに対しても行うことができる．例えば，ソフトウェアの開発に関しては，ソフトウェアだけでなく，開発プロセスにも適用できる．ハザード分析手法 STPA (System - Theoretic Process Analysis)， 事故分析手法 CAST (Causal Analysis based on STAMP)， STPA-Sec (STPA for Security)， STECA (System-Theoretic Early Concept Analysis) といった分析法が提案されており，航空宇宙分野をはじめセキュリティや医療など様々な分野ですでに利用されている．

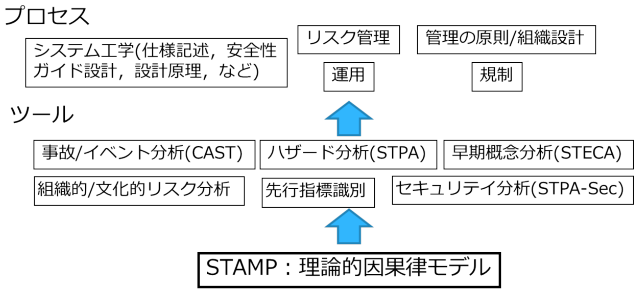


図 4. システム理論的因果律モデル STAMP とその活用プロセス

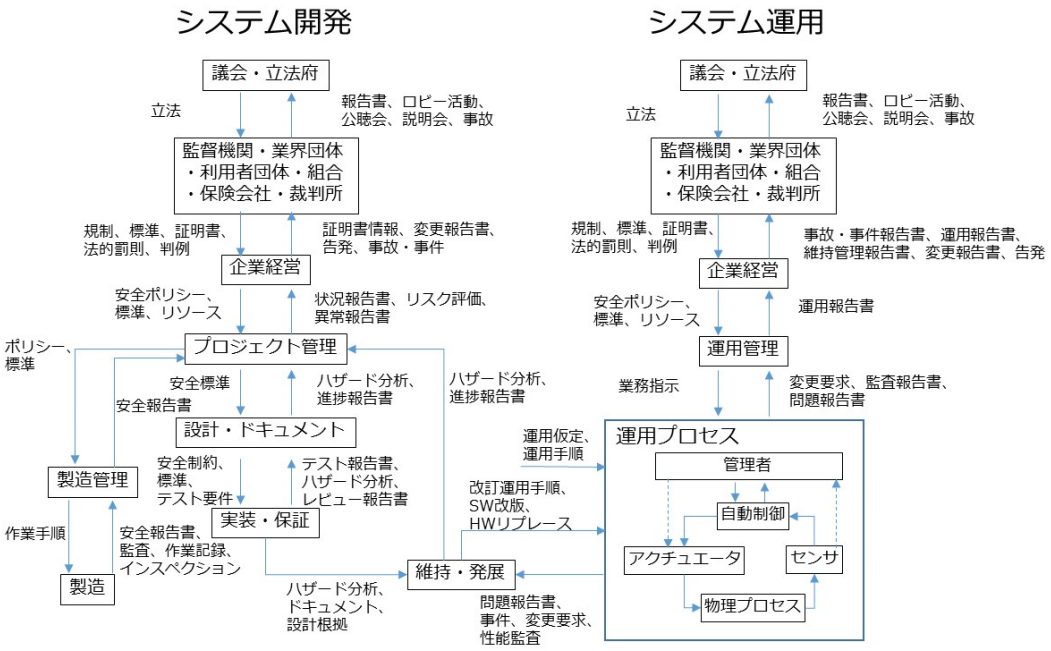


図 3. 開発と運用におけるコントロールストラクチャの例

ソフトウェア開発プロセスの場合，プロセスを設計する場合は STPA，実績データの分析に基づく改善の場合は CAST を適用するといった使い分けが考えられる．

2.2. STPA

STPA は，STAMP モデリングにおける代表的なハザード分析法で，典型的にはシステム開発の際にトップダウンで用いられる [7]．STAMP/STPA では，まず対象システムを理解するといった準備の後，回避すべきアクシデントとハザードを決めて STAMP によるモデリングを行い STPA の分析に着手する．STAMP モデルの前述の三つの要素を用い，コントロールを行う側とその対象プロセスとの間の相互作用において，安全制約が不適切であったり，守られない状態になったりするシナリオを中心に分析する．

以下に，STPA の手順の例を示す．実際には，ハザード分析を行う目的や段階の違いも含め，STAMP/STPA の具体的な適用法には様々なバリエーションがあり得る．

- 準備 1 ： アクシデント，ハザード，安全制約の識別
- 準備 2 ： コントロールストラクチャーの構築
- Step1：安全でないコントロールアクション（Unsafe Control Action：UCA）の抽出
- Step2 ：非安全なコントロールの原因の特定

最初に，アクシデント，ハザード，安全制約を決めるが，STAMP/STPA の特徴の一つとして，安全工学の技法を広い分野で適用できるようにとの意図の下、これらが定義されていることがある．プロセス改善の場合，プロセスに関する受容できない損失をアクシデントとすることで，STAMP/STPA が適用可能である．

アクシデント，ハザード，安全制約，コントロールストラクチャを定めたのち，コントローラからのアクションのうち以下の 4 つのタイプの非安全なコントロールアクションを識別する．

1. Not Providing causes hazard（与えられないとハザード）：安全のために必要なコントロールアクションが与えられないことがハザードにつながる．
2. Providing causes hazard（与えられるとハザード）：ハザードにつながる非安全なコントロールアクションが与えられる．
3. Too early/too late, wrong order causes hazard（早過ぎ，遅過ぎ，誤順序でハザード）：安全のための

ものであり得るコントロールアクションが，遅すぎて，早すぎて，または順序通りに与えられないことでハザードにつながる．

4. Stopping too soon/applying too long causes hazard（早過ぎる停止，長過ぎる適用でハザード）: (連続的，または非離散的なコントロールアクションにおいて) 安全のためのコントロールアクションの停止が早すぎる，もしくは適用が長すぎることがハザードにつながる．

ハザードにつながる 5 番目のタイプとして，必要なコントロールアクションが与えられたけれども実行されない，という場合もある．

非安全なコントロールを識別したのち，それにつながるシナリオを考え原因を識別する．安全制約を保つためのコントロールループやその構成要素を確認し，以下のような点を分析する．

1. プロセスモデルが正しくなくなってしまう原因も含め，どのようにして非安全なコントロール，ひいてはハザードにつながるかを分析する．
2. 必要なコントロールアクションが与えられたにもかかわらず，適切に実行されない原因を分析する．この分析結果は，(FTA, HAZOP, FMEA といった) 既存の分析の結果を含み得る．

図 5 はコントロールループにおいて，齟齬の原因となりうるものの例である

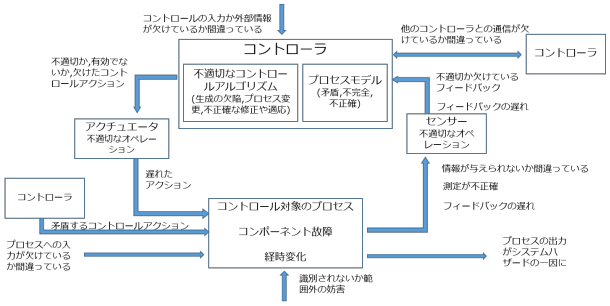


図 5. Causal Factor の例

3. プロセス改善フレームワーク PSP

ここでは個人レベルのプロセス改善フレームワーク PSP を簡単に紹介する．PSP は個人レベルのプロセスを対象としているが，CMM (CMMI の前身) に基づいて開発され，レベル 5 相当のものとして設計されている．そのトレーニングコースには複数のものがあるが，いずれも段階的にプロセス改善の知識とスキルを獲得する．図 6 に 8 つの課題に取り組むコースの概要を示す．

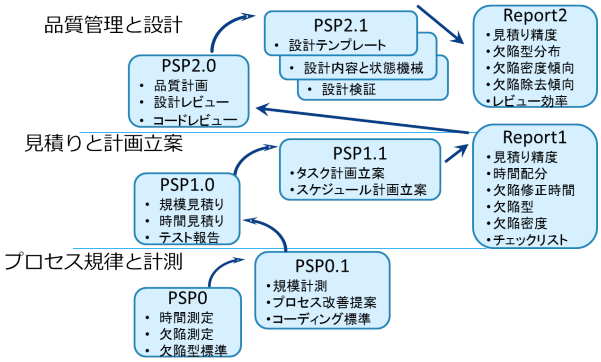


図 6. PSP トレーニングコースでの段階的な知識とスキルの獲得

知識とスキルに関する講義を受け、対応する演習を行う。その規模や欠陥等のプロセスの履歴データに基づき，プロセスに関する自身の問題点を定量的に分析する．分析結果に基づき，プロセス改善提案を行い，引き続く演習の実績により改善効果を確認する．受講者は，このような一連の講義と演習とを通じて，高品質ソフトウェア開発のための継続的な改善に必要とされる，プロセスの知識とスキルの定着を目指す．

このような段階的なプロセス改善を，STAMP モデリングに当てはめると，図 2 のコントローラ中のプロセスモデルに含まれる状態変数や，それらを監視・制御するアクション数が増え、アクションを決定できるアルゴリズムが高度化することに対応する。

4. PSP への STAMP 適用

4.1. PSP のインストラクタ向け分析

ここでは，PSP トレーニングコースのインストラクタが適切に受講者を指導するための分析を STAMP モデリングを用いて系統的に行うことについて議論する．各開発者個人が PSP トレーニングコースを受講し，必要な知識とスキルが定着した後は，図 1 でモデル化される

ような継続的な改善活動を個人レベルで行うことが想定される．しかしながら，トレーニング中は，図 7 のような入れ子のモデルとなる．受講者が各ステップでの知識とスキルの確実な習得を目指した活動を行うと同時に，インストラクタは受講者を外側から観測し各ステップでの確実な知識とスキルの習得を促進する指導を行う必要がある．

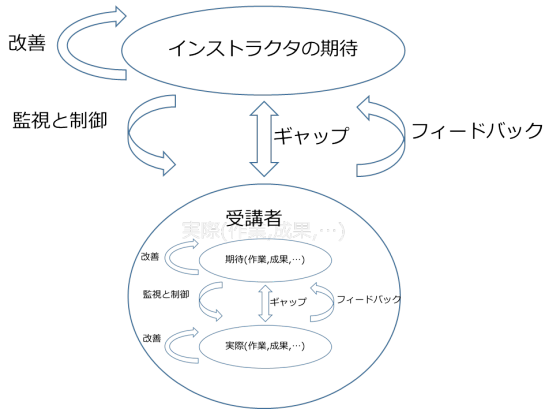


図 7. PSP トレーニングコースにおけるインストラクタの期待と受講者の実際

これを図 2 のような STAMP のモデリングに当てはめて考えると，インストラクタは，トレーニングの進捗に応じた受講者の期待モデルを持つと同時に，受講者の実際の状態を適切に把握するフィードバックを得る仕組みと，指導のアクションが出せるコントロールアルゴリズムを持つ必要がある．このような STAMP モデルに STPA 分析を行うことで，PSP トレーニングでのインストラクタの振る舞いに関するハザード分析が可能となる．

例えば図 8 に示すようなフェーズ構成を持つ PSP2.x では、インストラクタが受講者の状態のフィードバックを得て指導を行うタイミングは、典型的には計画フェーズと事後分析フェーズが想定されている．その際のインストラクタの振る舞いに対して STPA 分析を行うことで，指導方法を検討する際の系統的な分析ができる．指導の失敗事例に対して STAMP モデリングを行い失敗の原因について CAST 事後分析を行うことで，失敗事例を系統的に分析できる．

文献 [5] では，動機づけに着目した状態と操作により動機づけプロセスを定式化し，PSP 教育手法の改善への応用を論じているが，動機づけ失敗も含めたシナリオを系統的に分析できているわけではない．我々のアプローチでのプロセス改善に関する状態変数の集合に，動機づけに関する状態変数を加えることで、動機づけも含めてたモデルで分析できる．トレーニングコースの受講者は，通常は研究室やプロジェクトに属しているなどで、動機づけに関する他のコントローラが存在していることが多い．図 5 のように，受講者にとっての複数のコントローラを明示的に分析に加えることで、インストラクタ以外からの動機づけへの影響も含め系統的に分析が可能になる。

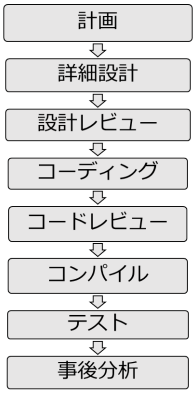


図 8. PSP2.x でのフェーズ

4.2 STAMP による形式手法を用いたプロセス改善分析

PSP では，プロセスデータに基づく改善を可能とするような測定の仕組みが埋め込まれており、履歴データに基づいてプロセスをハザードに導くシナリオを分析できる．例えば、主要な履歴データの一つである欠陥の記録には、自由記述的な欠陥の説明に加え、欠陥のタイプ、混入したフェーズ、検出・修正したフェーズといった事前に定められた区分に基づく記述も含まれる。このような欠陥の記録に基づいて、望ましくない欠陥が生じている状態をハザードとして，混入が特に多いフェーズや、混入から検出までのフェーズ間距離が長い欠陥のタイプといった観点で分析に着手できる。

PSP の各フェーズの進行手順には、開始基準や終了基準が設けられているが、それらが適切に守られていないことがハザードに結びついている可能性がある。例えば、特定のフェーズで欠陥の混入が多発するというハザード

の場合、フェーズ内での作業内容の問題に加えて、開始基準やその確認法といった問題で、本来はまだ開始できる状態にない作業に着手している可能性がある。特定の欠陥タイプに、混入から検出までのフェーズ間距離が長いものが多い場合、そのパス上のフェーズの終了基準の内容や確認法に、該当欠陥タイプ固有の問題があり、本来は検出されるべきところで欠陥が検出できずにすり抜けている可能性がある。

このような状況を STAMP のコントロールストラクチャに対応付け、ハザードが発生するシナリオを分析し、例えば以下のように形式手法導入の有効性を分析できる。

- 作業に関するアクションの結果である成果物に着目し、フェーズ間で受け渡される作業成果物の曖昧さの影響の伝搬を防止するため、形式的な仕様記述言語で作業成果物を記述する.
- 作業の進行を適切に制御する観点で、開始・終了条件の厳密な判定に焦点を当てる。あるフェーズで準備が不十分なまま作業が開始されているのは、前フェーズの終了基準の判定が不適切であった可能性も考慮する。問題が顕在化しているフェーズの前フェーズの終了基準を厳密に判定するために形式的な仕様記述の導入を検討する。

5. おわりに

システム理論的因果律モデルを用いてプロセス改善を分析するアプローチを提案した．本稿では主に個人レベルのプロセス改善を対象に、アドホックに行ったこれまでの事例に，提案手法を適用し，同様の改善をより系統的に導くことができる見通しを得た．これらの結果をふまえ，プロセスのテーラリングや改善についての系統的に取り組を発展させていくことができると考える．

本稿では主に個人レベルのプロセス改善を論じたが、STAMP は解析的還元論で説明できない創発的な特性も対象にできるため、チームレベルのプロセス改善も分析できる。図 3 に示すように、STAMP では，人や組織間の関係，規則まで含めた要因分析も可能であるため、CMMI のような組織レベルの分析も対象とできる。これまでの研究で、プロセス改善モデル CMMI-DEV[8] のモデル要素を用いた分析で，形式手法のような要素技術を起点にしたアプローチであっても，直接関係する技術的なプロセス領域以外の様々なプロセス領域との関係も考慮する必要があることが確認できている [9]．これらの結果をふまえ，組織レベルでの様々なプロセス領域に属するプラクティスや作業成果物の間の相互作用を STAMP モデリングを用いて分析する研究を今後行う予定である．

参考文献

[1] Humphrey, W. S.: PSP: A Self-improvement Process For Software Engineers, Addison-Wesley Pub, 2005.

[2] Team Software Process, http://www.sei.cmu.edu/tsp/

[3] 小材健，日下部茂，大森洋一，荒木啓二郎：測定可能な個人プロセスを対象とした形式手法導入に関する提案，情報処理学会ソフトウェア工学研究会研究報告，2009-SE-163-21 pp.17-24, 2009.

[4] 山田真也，日下部茂，大森洋一，荒木啓二郎：ソフトウェア開発チームプロセス演習 TSPi へのモデル指向形式手法 VDM の導入事例，ソフトウェアシンポジウム SS2012 予稿集, 2012.

[5] 梅田 政信, 片峯 恵一, 石橋 慶一, 橋本 正明, 吉田 隆一,“ソフトウェアプロセス教育における動機づけプロセスの定式化と教育改善への応用,”信学技報, vol. 113, no. 71, KBSE2013-10, pp. 55-60, 2013 年.

[6] Nancy Leveson, Engineering a Safer World, MIT press, 2012.

[7] An STPA Primer, psas.scripts.mit.edu/home/wp-content/uploads/2015/06/STPA-Primer-v1.pdf

[8] CMMI, http://cmmiinstitute.com/

[9] 日下部茂，林信宏，大森洋一，荒木啓二郎，ソフトウェア開発プロセス改善モデル CMMI-DEV の関連プロセス領域ネットワークの中心性分析，日本ソフトウェア科学会 コンピュータソフトウェア, Vol. 32 (2015) No. 3 p. 3_126-3_136

新商品分野でのソフト品質保証プロセスの構築事例

久木 達也
株式会社リコー
tatsuya.hisaki@nts.ricoh.co.jp

要旨

本稿では，プロジェクションシステム事業（PJS 事業）の新規立ち上げに伴い、新たなソフト品質保証プロセスを構築した事例を報告する．

事業ニーズに確実に応えるために，従来からの品質保証プロセスを適用した場合の課題を洗い出し，個々の課題に対し，商品分野の特性や組織の実力を考慮しながら，商品開発／品質保証プロセスを検討した．

この活動の内容と実施結果，及び，得られた知見について述べる．

1. はじめに

弊社では，2010年頃から従来からの基盤事業である複写機/複合機/プリンタ事業に加え，複数の新規事業の立ち上げてきた．

もともと，基盤事業における商品開発／品質保証のプロセス／環境が存在しており，これを新規事業に適用することにより，ある程度安定した品質／生産性で開発できる状態になっていた．

しかし，新しい商品分野で同じプロセスを適用すると，プロジェクト目標に対して大幅な納期遅れが予測されたため，商品分野の特性に合わせてプロセスを見直すこととした．

プロセスの見直しに当たっては，従来の組織の役割から見直す必要があるため，関係部署（PM 部署，品質保証部署，設計部署）が合同で検討にあたった．

2. プロセス見直しの方針について

基盤事業では，多機種並行開発を行っており，既存プロセスは，プロセスに応じて組織を編成し，各組織が最高のパフォーマンスを発揮できることを意図して設計されていた．新規参入したばかりの PJS 事業では多機種並行開発に最適化されたプロセスは重く，適さない．

そこで，品質を阻害しない範囲で，可能な限り品質保証を前倒しすることで，TAT 短縮を目指した．（図1）

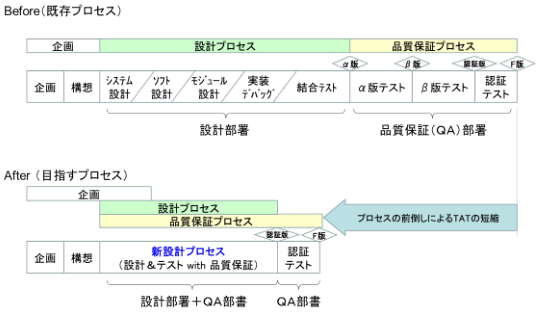


図1 プロセスの見直し

これを実現するために3つの改善を行った

① 品質保証の早期開始
② 設計プロセスの効率化
③ 段階的なリスク解消

（これら施策の詳細についてはスライドを参照．）

3. 実施結果

1機種目では，多くの課題は残したものの，当初の目的であった納期短縮は達成できた．現在まで機種の世代を重ねる度に改善することで，品質と生産性が安定してきており，基盤として定着したと評価している．

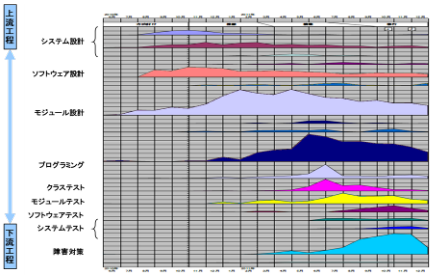


図2 開発工数の推移

4. まとめ

組織や新しい商品分野の特性に合わせた品質保証プロセスを構築する方法に関する事例を述べた．

情報システム開発におけるソフトウェア資産の上流シフトへの対応

松山浩士
(株)サイバーリンクス
ko-matsuyama@cyber-l.co.jp

鯵坂恒夫
和歌山大学
ajisaka@sys.wakayama-u.ac.jp

飯ヶ谷拓真
和歌山大学
s175002@sys.wakayama-u.ac.jp

要旨

プログラムコードを自動生成するアプリケーション開発ツールが普及するきざしがある. このツールはデータベーステーブルの設計も自動化するので, ソフトウェア開発のプロセス・視点・方法論がこれまでと大きく変わる. プログラムもテーブルも資産(asset)ではなくなり, 問題記述・要求仕様のパターンや記述法がそれに代わることになる. 本稿では, このような上流資産として望まれる性質を求めるための試みについて報告する.

1. はじめに

かつて高級言語が当たり前のものになってしばらくした頃, 超高級言語(VHLL)ないし第4世代言語(4GL)によるプログラム自動生成ツールが提案されたが, 以来約50 年間, 目立って進歩することはなかった. その理由は, 生成ツールの入力となるべき記述を個々のアプリケーションの要求仕様にきめ細かく対応させることや, 生成されるプログラムの実用的効率を確保する設計の自動化が難しかったことである.

現在に至って, これらの困難が解消される状況が整ってきた. コンピュータとネットワークのハードウェア性能が劇的に向上したために, 機械的に生成された結果をそのまま受け入れ, 性能を引き出すために込み入った設計を加えなくても, 利用品質として問題なくなったことによる. 特殊な独自仕様を与えられても, 高次正規形のテーブルに対するデータアクセスロジックをそのまま実装してもよくなった.

さらに情報システムについては, もうひとつ過去と異なるここ十数年の事情がある. すなわち, ウェブ, データベースやユーザ認証など, サーバ連携によるシステム構築が通例化したことである. それにともなって開発言語が単一のプログラミング言語だけではなくなり, 種々のスクリプト言語やフレームワークを調整するコンフィギュレーションファイルの記述など, 広範な技術スキルが求められるようになったため, 個々のシステムごとにいわゆるスクラッチでこれを開発していては, 求められる生産性と品質に対してコストが見合わなくなってきた.

このような状況を見計らって, データベースを扱う情報システムを対象とする新世代のアプリケーション自動生成ツールが登場してきた. これを活用したシステム開発法は当然ながら従来の方法論とは異なる. モジュールあるいはクラス設計を核とする考え方にとらわれていたのでは, むしろツールの強みを阻害することにもなりかねない. プログラムやデータテーブル設計を資産として活用し続けようという考えも改めたほうがよい. 問題の記述, 要求の仕様化に資産がシフトするので, より高い資産価値のあるものがどのような視点と手法によって得られるか明らかにすべきである.

この目標に向け, 本研究では具体的なツール事例として ”GeneXus”[1]を取り上げ, 目的とするシステム記述の内容とプロセスを確認する. 開発の重要なポイントがどこからどこへ移るのかを考察する.

2. GeneXus の概要

ひとつの開発プロジェクトに必要な情報群を集めたものを GeneXus ではナレッジベースと呼んでいる. ナレッジベースごとに Web アプリケーションや Windows アプリケーションなどの種別, ターゲット・プログラミング言語の指定などができる. ナレッジベースには 9 種類のオブジェクト(記述フォーム)を作成するが, とくに重要なトランザクション, Web パネル, プロシージャの3つについて以下に概説する.

2.1. トランザクション

トランザクションは, 業務上ひとまとまりの作業となる一連のデータ操作を定義する. まず, タプルデータの属性を決め, 個々の属性および属性間の規則を記述する. これらの情報をもとに GeneXus は正規化テーブルを生成する. 加えて, データの登録・変更・削除の画面が生成される. なお, 生成された画面に直接手を加えることもでき

るが，その時点でデータや操作の定義との関係が絶たれ，以後の変更に対する自動生成ができなくなる．

2.2. Web パネル

画面設計（トランザクションの記述により自動生成される画面以外のもの）のための定義記述群である．データの配置，受け渡し規則，システムおよびユーザからのイベント（コントロール）などが定義できる．

2.3. プロシージャ

データベースを読み出し，それにしたがってなんらかの出力（画面表示，帳票等印刷，ファイル書き込み）を作成するためのオブジェクトである．したがって，最も手続き的であり，出力レイアウトやパラメータに関する規則・条件を宣言的に書く部分もあるが，ソースコード記述が中心となる．

3. 開発重点のシフト

プログラム自動生成ツールを使うのであるから，プログラム論理設計・モジュール設計やプログラミングは，一部を残して大方なくなることは明らかである[2][3]．それ以外の大きな変化を二つ挙げて考察する．

3.1. データモデリングの上層シフト

データベースの三層スキーマは，上位から順に外部-概念-内部あるいは概念-論理-物理と呼ばれるが，データモデリングの作業はこれまでその中間層のスキーマ設計が中心であった．関係モデルではテーブルとタプルの設定であり，キー制約や正規化を十分に検討することであった．自動化ツールはその作業を代行する．したがって，重点は最上位層にシフトする．データモデリングがなくなり概念モデリング（外部スキーマ設計）に移行するといってもよい．

そこで記述するのは非正規表（樹形データ）でよい．属性値例を入力することにより反復群 (repeating group) が発見され，正規化された表が生成される．ここで再度，属性値入力をすることにより，正規化がさらに進むか完了かが判断される．

現実をより自然に（直感的に）反映するのは表より木であろう．正規表現の集まり帰着させるこれまでのデータモデリングは，全体が先にあり（定められていることが必要であり）それを部分に分ける分解指向の設計であった．外部スキーマ／概念スキーマを非正規表（樹形データ）で表現する概念モデリングはこの逆である．部分を集めて（成長し続ける，そのときどきの）全体を構成する統合指向システム構築法である．

概念モデリングにあたっては，システムに必要な概念群の全体を最初に捕捉しておく必要もない．例えば図書館システム例題について，貸出業務，棚返却業務，取寄せ業務など（関連するが）異なる業務に必要な画面・帳票のデータ定義は，どの順番で行っても最終結果は同じになり，また一部の定義でとどめても部分システムとして正常にはたらく．次々と現れる画面・帳票のデータ同定を繰り返すことでいわゆるインクリメンタル開発が自然に実現する．

SQL や DAO を含むプログラムはこれまで資産価値の高いものであったが，テーブル設計の必要性がなくなるとともに，それらは破棄されざるをえない．

3.2. 画面・帳票の自動化

ソフトウェア工学の教科書的な記述にはあまりみられないが，開発現場では画面・帳票の設計からスタートしフィックスするという慣例が根強くあった．それが業務上のまとまり作業を認識させるからである．従来はそれをもとに中間層のスキーマ設計をしていた．これは情報システム開発のハイライトのひとつであり，ソフトウェア/データベース技術者はこのスキルを手放したがらない，あるいはデータ中心の発想から抜けきれないかもしれない．

しかし GeneXus を用いた開発では，画面・帳票は設計対象ではなく，むしろ要求獲得の源泉となる業務的ドキュメントであり，それをもとにトランザクションを定義するという位置づけになる．一枚一枚の画面・帳票から起こされたトランザクション定義は，中間スキーマ設計を意識することなく，いかようにも積み重ねることができる．

先述のとおり，生成された画面には触れさせないというポリシーは，概念構造とトランザクションの構築に集中させるためとも考えられる．ユーザビリティや視認性という品質指標はたしかにあるが，属人性の強いものであるので，最大公約数的なところで折り合いをつけよう，エンタテイメントではなく業務なのだからそれでよかろうという姿勢だと思われる．

4. GeneXus を使った開発事例

データセンターにおける障害管理システムを事例として GeneXus（以下 GX）の活用について述べる．

4.1. 開発背景と目的

システムの障害が発生するとワープロソフトを使って障害報告書を作成している．報告書は障害の内容や経緯，原因や対策を書き管理者に報告される．報告後も根本原因の調査が続行される．その過程で発見される複数の原因や，個々の障害を誘発した誘因の連鎖関係を障害の根本対策が完了するまで持続的に管理しなければならない．根本対策が完了せず並行して管理しなければならない障害が増えてくると，個々の進捗状況を一括で管理できる障害管理システムが必要となる．

4.2. 開発方針

障害管理業務プロセスは，「障害管理作業」－「問題管理作業」－「変更管理作業」－「リリース管理作業」がある．現状はワープロソフトや表計算ソフトを使って場面ごとに報告書や一覧表など業務ドキュメントを作成し作業している．今回の開発は，前述した開発範囲をあえてはじめから意識せず，障害管理作業で使われている「障害報告書」の作成システムを開発することからはじめる．その後一つ一つの作業をシステム化し，最終的に障害管理業務全体のシステムの開発を目指す．

4.3. システム開発

4.3.1. 障害報告作成システムの開発

障害が発生すると責任者は障害報告書(図1)を作成し管理者に報告する．この報告書を作成するシステムを開発する．

図1　障害報告書

まず，障害報告書の属性を表の列に展開する．その後，属性値例を入力し反復郡を発見しては表の左端に詰めながら列の順番を置き換えていく．また，明細を発見していく（図 2）．

図2　障害報告書非正規表

次に GX のトランザクションを定義する．非正規表で発見した反復属性はトランザクションの外部化をおこなう．また，明細属性はレベルを下げ定義する（図3）．

コンパイルすると正規化されたテーブルとデータアクセスロジック，画面が生成される（図4）．

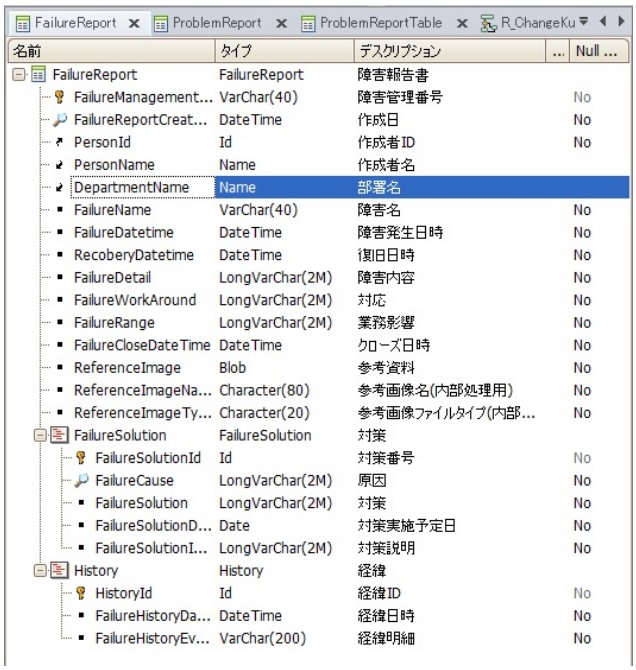


図3　障害報告書トランザクション定義

図4　自動生成された障害報告書作成画面

生成されたテーブル構造は図5の通りである．外部化した属性と明細指定した属性が正規化されテーブル生成されているのがわかる．

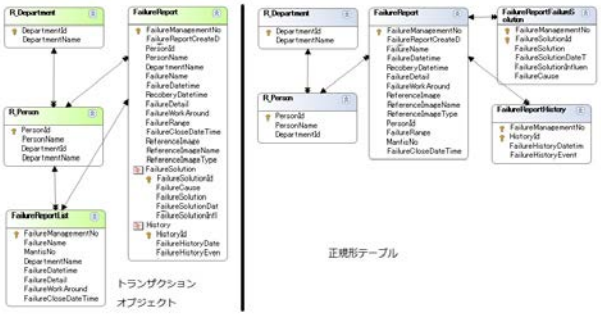


図5　障害報告書システムの正規形テーブル

4.3.2. 障害対策進捗管理機能の追加

障害対策の進捗を確認する為の一覧表を追加する．属性はほぼ障害報告書と同じで,「Mantis」という障害対応の記録が書かれている外部システムの管理番号が増えている(図6)．

部署名	障害管理ステータス	障害管理番号	Mantis管理番号	障害名	障害発生日時	障害内容	対応	クローズ日時
繰返	繰返							
クラウド基盤セル	問題管理	20151016-01	1563	紀三井寺データセンタ	2015/10/16 3時31分	紀三井寺データセンタ(本	① 新規接続ス	
		20151102-01	1447	海南DC仮想環境障害	2015年11月2日 10時23分	海南データセンタ内一般向	① 稼働してい	
	クローズ	20150910-01	1576	紀三井寺DC仮想環境	2015年9月10日 2時14分	紀三井寺データセンタ内仮	不調スイッチの	2015年12月31日 10時00分

図6　障害対策進捗管理非正規表

GX のトランザクションを追加する(図 7)．概念が同じ属性は既に稼動している障害報告書のトランザクション定義と同じ名前で登録する．そうすることで生成されるテーブルも同一概念として扱われテーブルと画面が自動生成される(図 8)．

図7　障害対策進捗管理表トランザクション定義

図8　障害対策進捗管理画面

4.3.3. 問題管理機能の追加

障害の原因と対策,そして緊急度とインパクトから導出した優先度を管理する機能を追加する．設計と開発の手順は先の二つと同様である．生成された画面とテーブルは次の通りである(図 9, 図 10)．

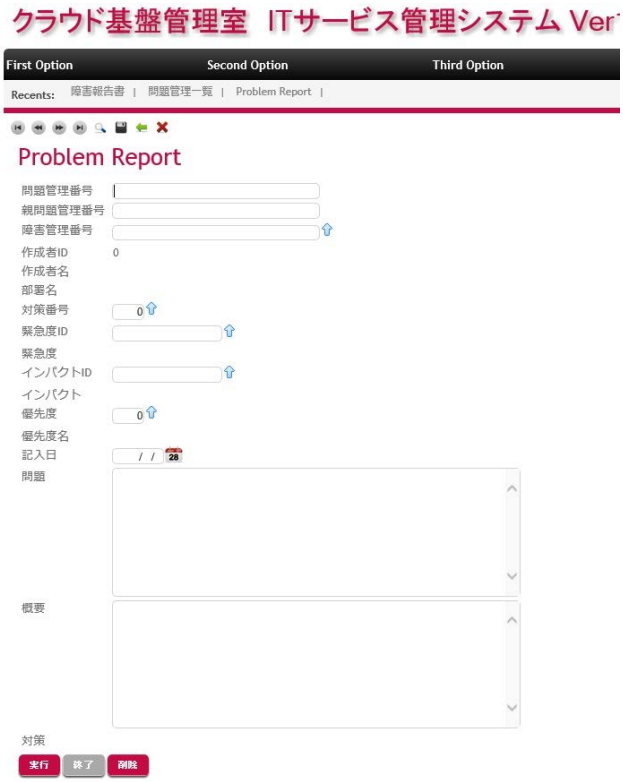


図9　問題管理画面

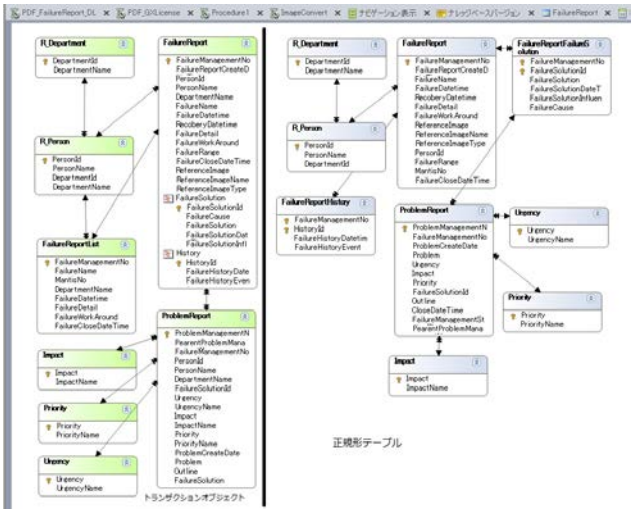


図 10　問題管理が統合された正規形テーブル

4.4. 考察

ハードウェア性能の向上により,機械生成された高次正規形のテーブルに対するデータアクセスロジックを利用してもパフォーマンスに問題がなくなり. 4 GL はそういった環境下で有用性を発揮できるようになった.

その結果プログラムやテーブル設計(データモデリング)に開発工数かからなくなり, ソフトウェアの資産性が上流シフトしている. それにもかかわらず開発現場ではあまりその意味が明確に認識されていない.

ソースコードが自動生成されることで,プログラミング工数が削減され, 単体テストや結合テストの工数も削減される. そのメリットは感じているものの, 開発手法は従来のままである. まずシステム開発の範囲を決め, 要求を洗い出し,データモデリング(テーブル設計)をおこない,そこから画面や帳票とのデータアクセスロジックを実装するというやり方だ. これは全体が先にあり(定められていることが必要であり)それを部分に分ける分解指向の設計である.

テーブルやデータアクセスロジックに変更があってもその都度自動で再編成できることで, プログラムの構造化, カプセル化, モジュール化, など, 流用性や保守性に動機づけられた技術へのニーズはなくなる. ソフトウェア開発の自動化がもたらす, 資産の上流シフトへの対応として, 部分を集めて(成長し続ける, そのときどきの)全体を構成する統合指向システム構築への思想的転換の可能性を明らかにした.

5. おわりに

プログラムが自動生成されるということは, 単体テストや結合テストもなくなるということである. しかし, システムテスト・総合テストはなくならない. もともと verification ではなく validation の性質をもつこの最終テストは, どのように計画すればよいのか. 自動生成ツールを使う開発と従前の開発では違うのかどうか, 今後の重要な検討課題である.

もうひとつの興味あるテーマは, 完全にはなくならないプログラム記述はどのような性質のものなのか, それをできるだけ宣言的な記述の解釈で実現することにどのような得失があるのかを精査することである.

さらに大きな問題は, 人間の思考的工数が上流へ集中できるようなった, 集中するしかなくなったときに, 様々に考えうる概念モデルのなかから, なぜあるひとつを選択

するのかという意図，理由 (rationale) を説明する方策であろう.

参考文献

[1] http://www.genexus.com/global/home?en
[2] 大澤文孝：GeneXus 入門，日経 BP 社（2012）
[3] ウィング：はじめての GeneXus，カナリアコミュニケーションズ（2011）

BRMS を適用するためのフィージビリティ・スタディ

― ビジネスルール設計方式と実行性能 ―

李 東昇
(株) 富士通システムズ・イースト
li.dongsheng@jp.fujitsu.com

要旨

経営やビジネス環境の変化に追随するため、情報システムの開発・変更を即応させなければならない。手段として、超高速開発[1] [4] [5] [6]という考え方とそれを支える超高速開発ツールが注目を浴びている。BRMS (Business Rule Management System) やノンプログラミングツールは、その代表である。BRMS の適用に向けて、下流工程での主な課題として「ルール設計方式の指針はどう決めるか」と「ルール方式導入による性能的に不安はないか」の２つがある。これらの課題に対してビジネスルールの組み合わせ方と非機能要件項目[2] (性能や保守性等) に着目しフィージビリティ・スタディを行った。本論文では疑似的な業務モデルを想定し、性能測定を通してルール設計方式の選択指針について述べる。

１．背景

経営環境の変化や技術の進化に伴い、情報システムに俊敏な対応が求められている。具体的には、短期開発やビジネスの変化対応力を実現するために、BRMS の適用が求められている。商談においても、RFP ((Request For Proposal) に BRMS 適用が条件となることが多く、今後の SI 技術には欠かせないツールとなってきている。

産業業界ではメインフレームからの脱却を目指しているお客様が増えている。これらのお客様は基幹システムのマイグレーションの検討に当たり、「現在の業務システムが将来に渡って継続的に運用できるか」と「プログラム仕様のスパゲッティ化によるメンテコストはどう削減できるか」という課題に直面している。解決手法として BRMS の導入が検討されている。

BRMS は、アプリケーションの業務ロジック部分を「ルール」として定義し、「ルールエンジン」により管理・実行されるシステム・ソフトウェアのことである[3]。「ルール」は業務ユーザーでも理解可能な形式で記述でき、記述した「ルール」はプログラムに自動変換され、アプリケーションから利用・実行できる。業務ロジックの改修が必要な場合には、ルールの変更のみで迅速に対応できる点がメリットである。

富士通グループでは、オープンソースツール (JBossBRMS) [4]にも取組んでいる。しかし実適用の実施例がないため、業務ルール実装方式の適用検討、基本的な使い方のノウハウ及び性能の基礎値を確認する必要があった。本論文では JBossBRMS を使いビジネスルールの組み合わせによりアプリケーションを開発する際に必要となる、使い方のノウハウ及び性能の基礎値を得るために行った検討とその結果について論述する。

２．課題

JbossBRMS を業務システムに安心して適用できるようにするためには、ＢＲＭＳの開発標準が無いため、各現場にて調査し、方式設計や業務適用指針の検討などを個別に行わなければならない。下流工程での BRMS 開発標準検討の一環として、まずもっとも重要な以下の２つの課題にフォーカスした。

課題１：ルール設計方式 (スプレッドシート方式／マスタ FACT 方式) の選択指針がない。

JBossBRMS のビジネスルールは、スプレッドシートでコード実装する方式と、マスタデータをマスタＦＡＣＴに展開してパターンマッチングする方式の、２種類の実装方式がある。どちらの方式が適切か、開発標準での観点からルール設計方式の方針を決めなければならない。

課題２：実行性能に関する定量的なガイドがない。

ルールの実行性能については定量的な評価を実施するため、サーバー上でほかのアプリケーションが動いていない“無風状態”と、トランザクションが集中している“高負荷状態” の２つの状態の性能を計測し、性能劣化の状況を検証する。また、メモリの配置目処はどのぐらいすべきかについて、ヒープサイズの Max 値を抑制し、必要最低限のメモリサイズを確認する。これらの計測値をまとめて、BRMS 導入時の定量的な参考値として提示しなければならない。

３．課題の対応策

郵便番号情報を検索する業務モデルを実装し、非機能要件項目(性能や保守性等)を検証することにより、ルール方式の選択方式を明確にする。具体的に、Ｗｅｂアプリケーションにルール呼び出しを実装したアプリケーション構成とする。ルールの呼び出しは Servlet から API で行い、ルールのステートフルセッションプールを使用する。トランザクションの発生は、テストツールとしてオープンソースの JMeter を使用して行う。

一般に業務データの操作パターンは作成・検索・更新と削除（CRUD）で定義することができる。BRMS は基本的に処理条件の複雑さに依存する処理となる。したがって DB の処理性能とは異なり、条件の複雑さによる検証が必要である。今回はルールを偏りなく一般化するため、検索のパターンを選定した。

ルールの設計方式について、スプレッドシートでコード実装した方式（以下、ルール Base と略す）と、マスタデータをマスタ FACT に展開してパターンマッチングする方式（以下、マスタ FACT と略す）の、２種類の実装方式を検証する。

図１は「条件項目＝10」の場合のルール Base の実装仕様のイメージで、「CONDITION」欄は条件項目で、「ACTION」欄は検索結果である。ルールのロード処理は、初回のみルール全体が Production Memory へ展開される。その後、ルールの実行はセッション取得、FACT Insert、Rule Fire とセッション解放 の順で行われる。

図２は「条件項目＝10」の場合のマスタ FACT の実装仕様で、マスタ FACT とトランザクション FACT をルールにより検索のためのパターンマッチングを行う。ルールのロード処理については、セッションプール取得時に重複したマスタロードを防止するため、プール内の FACT が空であれば、マスタの Insert を行う。マスタの Insert 方法として、アプリ側は一旦 ArrayList にマスタデータを格納後、それをルールエンジン側に渡す。そして、ルールエンジン側で ArrayList から抽出して Insert する。ルールの実行はルール Base と同じである。

RuleTable 郵便番号マスタ照会_1										
CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
$fc:照会データ										
照会 郵便番号桁 ==″$1″	″$1″ ==″0″	全国地方 公共団体コード ==″$1″	旧郵便番号5桁 ==″$1″	カナ都道府県名 ==″$1″	カナ市町村名 ==″$1″	カナ町域名 ==″$1″	都道府県名 ==″$1″	市町村名 ==″$1″	町域名 ==″$1″	$fc.set照会結果($1);
郵便番号7桁	備考5	全国地方 公共団体C	旧郵便番号 5桁	カナ都道府県名	カナ市町村名	カナ町域名	都道府県名	市町村名	町域名	町域名
0600000	0	01101	060	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｲｶﾆｹｲｻｲｶﾞﾅｲﾊﾞｲ	北海道	札幌市中央区	北一条東	true
0640941	0	01101	064	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｱｻﾋｶﾞｵｶ	北海道	札幌市中央区	旭ケ丘	true
0600041	0	01101	060	ﾎｯｶｲﾄﾞｳ	ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ	ｵｵﾄﾞｵﾘﾋｶﾞｼ	北海道	札幌市中央区	大通東	true

図１　ルール Base の実装仕様のイメージ（条件項目＝10）

・マスタFACT

```
public class 郵便番号マスタ{
    public String 全国地方公共団体コード;
    public String 旧郵便番号5桁;
    public String 郵便番号7桁;
    public String カナ都道府県名;
    public String カナ市町村名;
    public String カナ町域名;
    public String 都道府県名;
    public String 市町村名;
    public String 町域名;
    public String 削除フラグ;}
```

・トランザクションFACT

```
public class 照会データ extends 郵便番号マスタ{
    public String 照会郵便番号7桁;
    public String 照会結果; }
```

左記の2ファクトを、下記ルールでパターンマッチングする。

・条件項目＝10の場合

```
rule ″郵便番号照会 Rule"
  when
    $Fact : 照会データ( $pram1:照会郵便番号7桁, $pram2:全国地方公共団体コード,
                $pram3:旧郵便番号5桁, $pram4:カナ都道府県名, $pram5:カナ市町村名,
                $pram6:カナ町域名, $pram7:都道府県名, $pram8:市町村名, $pram9:町域名 )
    $Mast : 郵便番号マスタ( 郵便番号7桁==$pram1, 全国地方公共団体コード==$pram2,
                旧郵便番号5桁==$pram3, カナ都道府県名==$pram4, カナ市町村名==$pram5,
                カナ町域名==$pram6, 都道府県名==$pram7, 市町村名==$pram8, 町域名==$pram9,
                削除フラグ==″0″)
  then
    $Fact.set照会結果(true);
    retract($Fact)
end
```

図２　マスタ FACT の実装仕様のイメージ（条件項目＝10）

実装仕様とルールのロード処理を比べると、図１のルール Base の設計方式は実装しやすさのみならず、“ルールの見える化”も優れている。マスタデータがない限り、ルール Base の設計方式を選択すべきである。（保守性）

既存マスタ情報の活用を考慮すると、図２のマスタ FACT の設計方式はマスタ DB を容易に取り込むので、マスタ FACT の方が有効であると考えられる。（移行性）

ルールの実行性能に関しては定量的な参考値を明確にするため、“無風状態”と“負荷状態” の２つの状態の処理性能を計測し、処理性能のフィージビリティを検証した。結果として、性能要件としてのレスポンスタイム（3 秒以内）には問題がないことを確認した。但し、ルールの実行性能への影響要因等についてはいくつの考慮点がある。その中で特に BRMS 特有の特徴と GC（ガーベージ・コレクション）による性能への影響について考察する。（性能）

１）“無風状態”でのルールロード時間とルール実行時間の計測

JMeter から１多重で 20 回の HTTP リクエストを連続（応答があり次第、次の要求を即座に）送信し、レスポンスタイムを計測する。図３は、条件項目数＝４（ルール Base とマスタ FACT）のロード時間とルール実行時間を計測した結果である。

ルール Base の場合、ルールの読み込み性能は、母数が大きくなるほど１ルールあたりの読み込み性能も劣化する傾向が見られる（120,000 件の場合は 19.1 ミリ秒）。マスタＦＡＣＴの場合、マッチング用ルール（DRL 言語）のルールロード時間が約 5,000 ミリ秒かかるが、１件当たりの読込み性能は対象のデータ件数による違いは見られない。JBossBRMS は、小規模なルールでも読込みオーバーヘッドが大きいので、事前にセッションプールにロードすることが必要である。実行性能は、どちらでも今回検証したルール件数１２万件までにおいてばらつきはあるものの、顕著な劣化は見られなかった。

（実測値）

ルールBaseの設計方式

No.	ルール数	ルールロード（ミリ秒）	ルール実行（ミリ秒）	読込み時間/rule
1	10,000	18,289.40	2.08	1.8
2	30,000	82,603.40	1.45	2.8
3	120,000	2,286,594.20	2.91	19.1

マスタFACTの設計方式

No.	データ件数	ルールロード（ミリ秒	マスタＦＡＣＴ展開（ミリ秒）	ルール実行（ミリ秒）	読込み時間/件
1	10,000	4,938.40	526.50	1.24	0.05
2	30,000	4,850.40	1,271.75	1.68	0.04
3	120,000	4,803.80	4,686.80	2.05	0.04

図３　“無風状態”の実行時間とロード時間

２）負荷状態でのルール実行時間の計測

3 万ルールが登録されている環境で、JMeter により HTTP リクエストを発生させて、「Rush 状態※」でのルールの処理速度を計測する。多重度は、50 多重、100 多重と 200 多重にして、それぞれ 75,000、150,000 と 300,000 のサンプルトランザクション数（多重度×500 リクエスト/回×３回）に対して計測結果を統計分析した。

※セッションプールの初期確保数を多重度分とし、このプールに事前にルールを読み込む為に、Rush 走行前に暖気運転として全スレッドから初回要求を送信し、その後にＲｕｓｈ状態となる。

図４は、ルール Base とマスタ FACT の「Rush 状態」でのルール実行時間とその分布（１ミリ秒未満、20 ミリ秒未満とそれ以外）を計測・統計した結果である。ルール実行時間は、セッション取得 ＋ FACT Insert ＋ Rule Fire ＋ セッション解放 の一連の処理の合計時間を指す。

（実測値）

ルールBaseの設計方式

No.	テストケース	ルール実行（ミリ秒）	1ミリ秒未満	20ミリ秒未満	それ以外
1	50多重	13.84	58.1%	20.1%	21.8%
2	100多重	23.75	57.9%	11.6%	30.5%
3	200多重	32.71	61.5%	7.7%	30.7%

マスタFACTの設計方式

No.	テストケース	ルール実行（ミリ秒）	1ミリ秒未満	20ミリ秒未満	それ以外
1	50多重	9.15	71.6%	14.6%	13.9%
2	100多重	14.03	68.4%	12.8%	18.8%
3	200多重	19.20	67.4%	10.4%	22.2%

図４　“”負荷状態”のルール実行時間と分布

ここでは、1,000 ﾐﾘ秒を閾値として、これを越えるデータを GC（ガーベージ・コレクション）による例外データとみなして統計から除外する。ルール Base の検索時間は 50 多重で約 14 ミリ秒と高速を維持して、200 多重でも約 33 ミリ秒という 2.3 倍ほどの数値に落ち着いている。また、マスタ FACT の検索時間がルール Base の約６割で、ルール Base よりマスタ FACT の処理性能が良かった。これで、負荷状態でも両方式とも良い性能を維持することが可能であると確認できる。時間と TPS を計測・統計した結果である。

図５は、200 多重において、Ｒｕｓｈ状態での経過時間(横軸：秒)と、その時点で受信した処理のルール実行時間（縦軸：ミリ秒）の分布図である。1,000 ミリ秒以上のデータは GC による例外データだと考える。

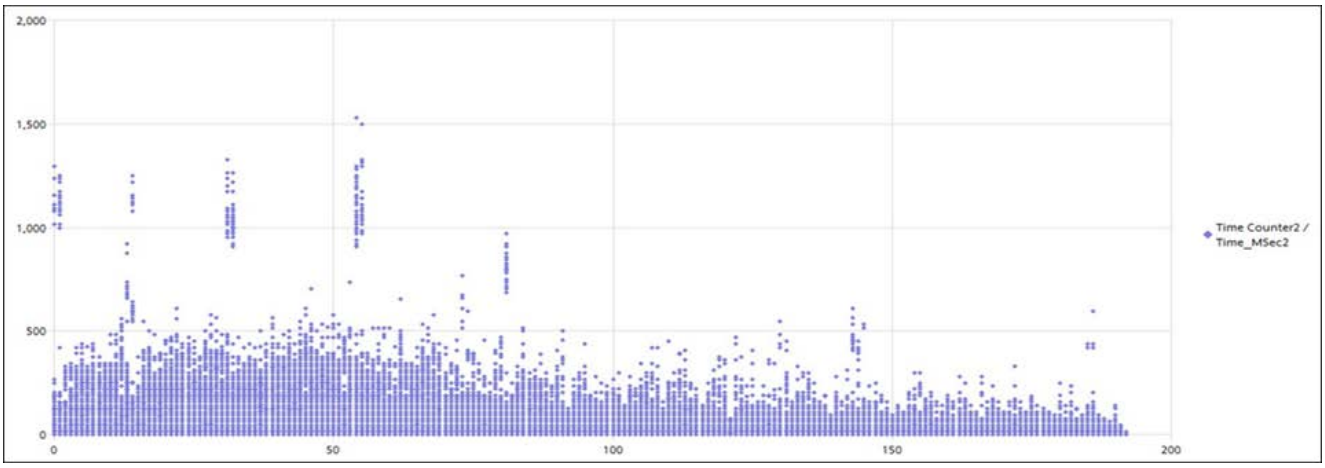


図５　２００多重の経過時間とルール実行時間の分布図

３）セッションプールの MAX 値と Idle 値による影響分析

セッションプールの作成時間は今回の計測では意識するレベルの時間は要しなかった。ルール Base の場合はルール全体で初回のみ Production Memory へ展開され、各セッションプールからは共有領域として参照されるため、Idle 分で起動されるセッションプールだけでなく MAX 値で新規展開されるセッションプールにおいてもルールロード時間は考慮しなくてよかった。マスタ FACT の場合は、アイドル状態のセッションプールは Production Memory の内容を維持できるので２回目以降は高速で処理できるが、Idle 値を超えた MAX 値までのセッションプールは毎回新規展開とセッションプールのクリアが行なわれるせいで、毎回マスタ FACT の展開分をロスすることとなった。環境としての資源が許されるなら、セッションプールの設定は Idle 値と MAX 値を同一にすることを勧める。MAX セッションプール数を超えるリクエストが集中した場合には、セッションプールの空き待ちによるロスが発生する。表１は、ルール Base　３万ルールでの「待ち無し」と「待ち有り」のセッション取得時間の比較であり、「待ち有り」は「待ち無し」の約 35 倍であった。この結果から、可能な限り、最大同時接続数をセッションプール数として確保できる方が望ましいと言える。

表１　**セッション取得時間の比較**

[セッション取得時間]　　単位:ミリ秒

No.	テストケース	①待ち無し	②待ち有り	①:②
1	50多重	0.86	29.94	1 ： 34.81
2	100多重	1.46	58.13	1 ： 39.82
3	200多重	2.41	87.30	1 ： 36.22

※待ち無し･･･起動時に多重数と同数のセッションプールを確保
待ち有り･･･最大セッションプールを10に制限して起動

４）GC による影響分析

“負荷状態”でのテストと同等の条件下で、ヒープサイズを抑制し必要最低限なメモリ容量を検証した。またその状況下での処理性能を測定した。検証した結果としてヒープサイズの最低必要量は、ルール Base の場合約 1.5GB、マスタ FACT の場合約 7GB であった。それを下回ると OutOfMemory Exception が発生する。

表２はルール Base　３万ルール、200 多重のテストケースに対して、ヒープサイズを抑制なしの 40GB、抑制ありの 13GB、抑制ありの 1.5GB、の３つの場合におけるルール実行時間と TPS を計測した結果※である。抑制ありの 1.5GB で計測した場合、かなりのＧＣが発生しており、TPS が下がった大きな原因になると言える。

表２　メモリ抑制ありとなし場合の“負荷状態”でのルール実行時間と TPS

テストケース	項目	抑制無し	抑制有り(13GB)	抑制有り(下限)	無 : 13GB : 下限
ルールBase 3万ルール200多重	YGC回数	11	20	140	1 : 1.8 : 12.7
	FGC回数	0	1	3	0 : 1 : 3
	ルール実行時間	32.71	24.20	19.24	1 : 0.7 : 0.6
	TPS	521	457	384	1 : 0.9 : 0.7

※ルール実行時間（平均値）を計算時、1,000ミリ秒を閾値として、これを越えるデータをGCによる例外データとみなして集計から除外する。

４．ルール設計方式のガイド（以下、本ガイドと略す）

１）ルール設計方式の選択指針

図６は、ルール Base 方式とマスタ FACT 方式に対する一部の非機能要求項目を比較したものである。保守性はルール Base 設計方式の方が優れている。移行性とルール管理の効率化はマスタ FACT のほうが有利である。ルール設計方針の選択例として、ルール数が少ない場合は可視化の良いルール Base が効果的と考える。既存マスタを活用したい場合はマスタ FACT 設計方式を選択すべきである。もちろん、複雑なルール設計の場合は、両方式を併用するハイブリッド方式も考えられるだろう。

	設計方式		
比較項目	ルールBase	マスタFACT	備　　考
性能	○	○	今回の測定したようにルールBase設計方式よりもマスタFACT設計方式の方が処理若干有効であるが、両方ともミリ秒のレベルであった。
保守性	○	△	“ルールの見える化”と実装しやすさはルールBase設計方式の方が優れている
移行性	△	○	・既存マスタ情報の活用 マスタFACTを設定する場合、他システムで利用中のマスタDBを容易に取り込むことが可能である
効率性	×	○	・ルール管理の効率化 コードから名称へ、逆に名称からコードへ変換するような双方向の変換が求められる場合に、ルールBase方式では双方向の変換ルールをそれぞれ記述する必要があるが、マスタFACT方式では一方向だけの記述で良い

図６　ルール設計方式の非機能要件項目の比較

２）ルールの実行性能の定量的な参考値及び考慮点

今回は主に３万ルールを基準として計測した。“無風状態”と“負荷状態”ともにミリ秒の実行性能なので、性能要件としてのレスポンスタイム（3 秒以内）には十二分な性能結果が出たことで問題なく活用できると考える。但し、ルールロード時間を減らすため、ルール実行サーバーでは事前にルール環境の立上げを進める。また、セッションプールに関するパラメータの設定や GC によって影響があることも確認できた。可能な限り最大同時接続数をセッションプールの MAX 値と Idle 値として設定することが望ましい。最後にヒープ領域をふんだんに使用するため、適正なサイズを確保しないと GC の影響を受ける。性能担保のためにも運用に合わせた、この領域のチューニングが重要である。

５．おわりに

超高速開発における BRMS の実用化について、JBossBRMS についての基本的な使い方のノウハウ及び性能の基礎値を確認することができた。これは JBossBRMS を顧客のシステムに適用する際の参考になりうる。ここ数年、BRMS についての問い合わせや相談などが急増している。これは、俊敏性をもとめた超高速開発への期待に比べ、一般書籍を始め、社内外に BRMS の適用に関する技術情報が少ない状況にあるためである。BRMS は、システムの一部かもしれないが、ビジネス全体から見ると重要度は高く、商談を発展させる切り札と考える。BRMS によるアプローチがビジネスを新しい方向へ導く鍵となる。今後ますます注目を浴びていく[5] [6] [7]と考える。

今回のフィージビリティ・スタディでは方式の使い分けと性能について検証した。しかし、実際の業務ルールのすべてを本当に表現することが可能か否か、ルール件数の量的面と業務の複雑さからの検証はまだ今後の課題として残っている。

参考文献

[1]「超高速開発のためのルールベース開発技術の研究」. LS 研成果報告書. 2014 年度
http://jp.fujitsu.com/family/lsken/activity/w

ork-group/14/abstract/outline_04.html
[2] 「経営に活かす IT 投資の最適化」. 情報処理推進機構（IPA）. 2011 年 4 月
http://www.ipa.go.jp/sec/reports/20110427.html
[3] 森田　武史、山口　高平.「業務ルール管理システム BRMS の現状と動向.「人工知能学会誌」 29(3), 277-285, 2014-05-01
[4] 「Red Hat Jboss BRMS」.
http://www.jboss.org/products/brms/overview
[5] 井上英明.「超高速開発」が日本を救う.「日経コンピュータ」. 2012 年 3 月 15 日号.
http://itpro.nikkeibp.co.jp/article/NC/20120309/385541/?ST=NC
[6] 井上英明.「みずほもすなる超高速開発」. ITpro By 日経コンピュータ. 2015 年 10 月 16 日
http://itpro.nikkeibp.co.jp/atcl/watcher/14/334361/082400358/?ST=system&P=1
[7] 井上英明.「広がる超高速開発　限界と常識を打ち破る」. ITpro By 日経コンピュータ. 2016 年 1 月 4 日
http://itpro.nikkeibp.co.jp/atclact/active/15/121700150/121700001/

性能トラブル解決の勘所

鈴木 勝彦
株式会社日立ソリューションズ
masahiko.suzuki.yh[at-mark]hitachi-solutions.com

要旨

ソフトウェア保守では，性能トラブルが発生すると原因究明までに時間がかかることが多い．これは，性能トラブル発生時の調査手順が確立されておらず，場当たり的で試行錯誤しながらの調査となっているためである．また，性能トラブルの原因としてどのような要素があるかなどが体系的にまとめられていないことにも起因している．

本研究では，性能トラブルの要素などを 4W1H の概念で体系化し，実際の性能トラブル発生時の調査の勘所をまとめた．性能トラブルでお悩みの方々のお役に立つことができれば幸いである．

1. はじめに

ソフトウェア保守では，ソフトウェアが稼働してからトラブルが発生することは避けられないが，原因究明にかかる時間は，原因によって短い時もあれば長期化することもある．中でも性能トラブルが発生すると，原因究明までに時間がかかることが多い．特に，解決までの時間は，担当者の経験，知識や技能によって全く違う．また，解決のノウハウがドキュメント化されていないため，ノウハウの伝承がうまくいっていない．著者は，一定の技術者であれば，性能トラブルが発生しても一定の時間で解決できる資料が必要と考えた．

当初は，著者自身が過去に経験した多くの事例集を作成した．しかし，ソフトウェア・メインテナンス研究会のメンバから単なる事例集では実際の性能トラブル発生時の調査では活用が難しいとのご指摘を受け，調査ポイントを列挙して段階的に調査できる形式に再作成した．しかし，すべての事例は網羅できないため，調査時の観点を体系化できないかを試行錯誤し 5W1H の概念が使えるのではないかと閃いた．最終的には，性能トラブルの要因などを 5W1H から「Where」と「Why」を除き，「Whom」を追加した 4W1H での体系化に至った．体系化したことで，過去に列挙した事例だけでなく，経験と体系を基に組合せることで，考えられる事例を追加し，より網羅性のある事例集を作成することができた．この論文を使用することで，一定のスキルを有する技術者であれば 4W1H の観点を思い浮かべながら，事例集を基に調査することで，解決までの時間を短縮することが可能と考えられる．

本論文では，最初に 4W1H による体系的な説明を実施し，最後にいくつかの性能向上施策方法を記述した．

2. 性能トラブル時の調査観点を 4W1H で考える

性能トラブルに対して，経験豊富な特別な技術者でなく，普通の技術者でも 4W1H による体系的な観点に基づいて調査すれば，試行錯誤することなく，解決までの時間短縮に有効であると考えた．

「4W1H」とした経緯は，体系的にするにあたって「5W1H」という考え方に基づき，過去 100 例ほどの性能トラブルに対して，When(いつ)，Where(どこで)，Who(誰が)，What(何を)，Why(なぜ)，How(どうやって)の6項目毎に観点を追加する作業を試みた．その際に，Why(なぜ)の項目は人の動機に該当し，物理現象には該当しないため除外した．さらに，外部影響によって性能トラブルとなる事例の分類の必要性から，Whom(誰によって)の項目を追加した．

また、Where の場所の特定はプログラムの場合には Who のプロセスの特定と同じ結果となり，Who に該当するプロセスに関する情報は，統計情報などで確認できるが，Where に関する情報は簡単には得られないため，Where も項目から削除し，結果として「4W1H」とした．一般的に情報伝達する時に「5W1H」を意識する必要があるように，性能トラブル時には，「5W1H」に対応する「4W1H」によって事象を正しく把握し，結果として原因究明ができるようになる．

「4W1H」のそれぞれについて，どのように捉えるかを以下に説明する．

・Who(誰が)は，原因となるプロセスを究明する．
　性能トラブルが Who(誰が)に該当するプロセスかを究明し，さらにそのプロセス内のどのソースコードかを究明する．

・How(どのように)は，原因となる動作を究明する．
　性能トラブルは，動作がスムーズに処理できていないため発生する．現象などから，どのような振舞いによって遅くなっているかを究明する．

・What(何を)は，原因となる資源を究明する．
　プログラムは，実行する時にさまざまな「資源」を消費しながら処理している．プログラムが，システムで供給できる「資源」を上回る消費を行おうとすると遅延が発生するため，不足している「資源」を究明する．

・When(いつ)は，原因となるきっかけを究明する．
　性能トラブルは，問題なく動いていても突然発生することがある．しかし，プログラムは，入力する情報が同じであれば，同じ結果になるハズである．遅延の発生は，ある時点(When)から入力情報が異なることを示すため，変わった時点を究明する．

・Whom(誰によって)は，原因を誘発したものを究明する．
　通常の性能トラブルは，遅くなっているプロセスに起因することがほとんどであるが，外部の影響を受けて発生することもある．このため，調査する時には自プロセスだけでなく，システム全体または他システムも含めて全体を把握する必要がある．

3. Who(誰が)は，原因となるプロセスを究明

性能トラブルが発生した時には，Who(誰が)に相当するどのプロセスに起因して発生しているかを特定する必要がある．プログラムは，1 つのプロセスだけでなくいろいろなプロセスが連携して動作している．また，1 つのプロセスの中でも自分が作成したソースだけでなく，同梱しているライブラリであったり，call している API などのソースもある．問題となるプロセスは，場合によっては自ホストとは限らず，他ホストに起因することもある．図 1 は，原因となるプロセスが Who(誰が)を分類したものである．

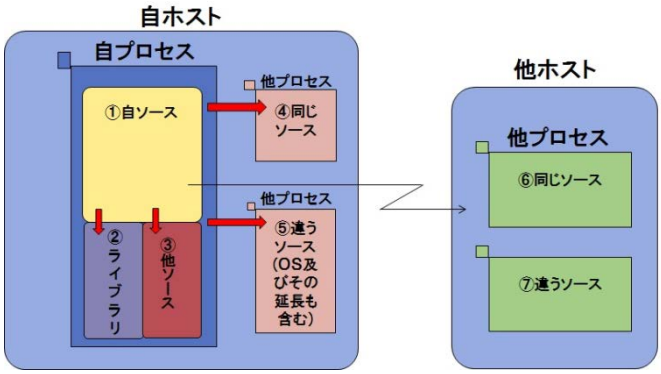


図 1.　Who(誰が)の原因となるプロセス究明時の観点

3.1　性能トラブルがどのプロセスで発生しているかの観点

性能トラブルが発生した時は，現象などから見た目の動作が遅いプロセスを特定するが，場合によっては他のプロセスが起因している可能性もある．実際には，プロセス単位の CPU 使用率，DISK 入出力回数と入出力バイト数，データ通信量などの統計情報などから問題のプロセスを絞り込んでいく．さらに，プロセスの中のどの部分で発生しているかを絞り込んでいく．Who(誰が)の絞り込みであるが，場所の特定という意味では Where(どこで)の要素も含まれている．

3. 2 原因となるプロセスの絞り込み方法について

原因となるプロセスを特定するには，最初にシステム全体及びプロセス単位での CPU 使用率，DISK 入出力回数と入出力バイト数，データ通信量を確認し，さらに次のような観点でプロセスを絞り込んでいく．

(1)自分のプロセスの CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，自プロセスの可能性が高く，次の 3 つが考えられる．

・自プロセスの自分で作成したソースで発生．
・自プロセスだが，取り込んでいるライブラリで発生．
・自プロセスだが，call している API の延長で発生．(システム関数の場合も含む)

(2)他プロセス(自分で作成したソース)の CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，他プロセスの可能性が高く，次の2つが考えられる．

・自プロセスの延長で他プロセスが実行されるが，他プロセスからの戻りが遅いことで自プロセスの性能がでないことがある．

この場合は，他プロセスに関して調査する．

・自プロセスは，他プロセスと通信しながら処理しているが，他プロセスからの戻りが遅いために性能がでないことがある．この場合は，他プロセスと通信の状態に関して調査する．

(3)他プロセス(他人が作成したソース)の CPU 使用率が高い，DISK 入出力回数が多い，または入出力バイト数が多い，データ通信量が多い場合は，他プロセスまたは，システム全体の可能性が高く，次の2つが考えられる．

・特定の他プロセスだけ(システムプロセスも含む)がリソースをたくさん使用し，自プロセスから call している延長で発生しているかを確認する．
・特定の他プロセスだけ(システムプロセスも含む)がリソースをたくさん使用し，自プロセスが確保するリソースと競合が発生していないかを確認する．

(4)システム全体の CPU 使用率が高い，DISK 入出力回数が多い，または転送バイト数が多い，データ通信量が多い場合は，他プロセスまたは，システム全体の可能性が高く，次のことが考えられる．

・システム全体が資源をたくさん使用している場合には，自プロセスが確保する資源と競合が発生していないかを確認する．
・システム全体のハードウェアのリソースが不足していないかを確認する．

3. 3 プロセスを特定した後，さらにどのソースコードで発生しているかを調査する方法について．

(1) 自分のプロセスでかつ自分のソースで発生

・自分のソースなので，トレースログを強化すれば，場所の絞り込みも可能．

(2)自分のプロセスであるが，取り込んでいるライブラリ部分で発生

・ライブラリの前後でトレースログを出力すれば，絞り込み可能．

(3)自分のプロセスであるが，API の延長(他人のソース)で発生

・API の前後でトレースログを出力すれば，絞り込み可能．

(4)他プロセスであるが，自分と同じソースで発生

・自分のソースなので，トレースログを強化すれば，場所の絞り込みも可能．

(5)他プロセスでかつ，他人のソースの部分で発生

・API の前後でトレースログを出力し，該当する他プロセスが自分の発行している API の延長(OS なども含む)であるかを確認する．
・他プロセスが特定の資源を大量に消費していないかを確認する．

(6)他ホストであるが，自分と同じソースで発生．

・通信処理の前後でトレースログを強化すれば，場所の絞り込みも可能．他ホストとの通信がある場合には，他ホストからのレスポンス待ちで自ホストの自プロセスが遅く見える場合がある．原因の究明は，自ホストと同様の方法で他ホストのプロセスを調査する．

(7)他ホストでかつ，他人のソースで発生

・通信処理の前後でトレースログを強化すれば，場所の絞り込みが可能．しかし，他ホストの他人のソースが起因する場合には，原因の究明は難しい．

4.　How(どのように)は，原因となる動作を究明

性能トラブルが発生した時には，How(どのように)に相当するどんな事象が起きているかを特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，分類すると大きく6種類になる．図2は，性能トラブルの事象の原因を How(どのように)の観点で分類したものである．

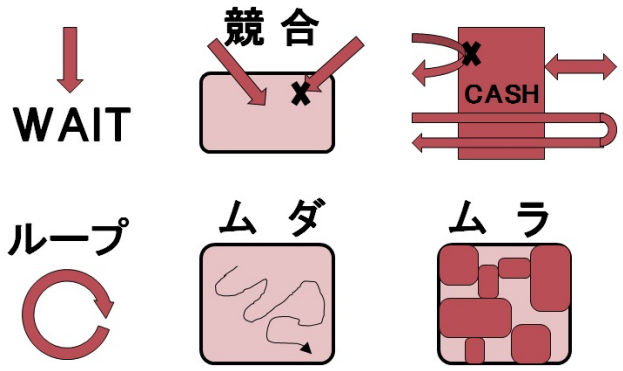


図 2. How(どうした)の原因となる動作を究明時の観点

4. 1 性能トラブルがどのようにして発生しているかの観点

3章では，性能トラブルがどのプロセスで発生しているかを究明する方法を記載したが，原因究明には，そのプロセスが具体的にどのようにして性能低下しているかを究明する必要がある．

性能トラブルの原因のメカニズムに関しては，大きく「WAIT」，「競合」，「キャッシュ」，「ループ」，「ムダ」，「ムラ」に分類できる．性能トラブル解析時には，この 6 つのどの原因で発生しているかを特定する必要がある．

4. 2 原因となる事象の絞り込み方法について

(1) WAIT による性能トラブルについて

WAIT(待ちが発生)は，性能トラブルの中でも一般的には原因究明が容易な方である．しかし，非常に短い WAIT が頻発している場合には，原因究明に時間がかかることがある．

排他資源の解放待ちは，「WAIT」と排他資源の「競合」の両方の要素がある．複数資源の排他によるデッドロックの場合には，自プロセスでわかる排他資源は 1 つであり，他プロセスの排他待ちで異なる排他資源を究明する必要があり，時間がかかることがある．

WAIT の具体的な要因としては，以下のものがある．

・タイマー値が大きい
・タイマー値が大きくリトライあり
・タイマー値が小さくリトライを多発
・相手の処理待ち
・無限待ち
・デッドロック(「競合」でもある)

(2) 競合による性能トラブルについて

競合は，ハードウェア資源の競合と，排他資源の競合がある．ハードウェアの競合は，リソースの状況を確認することで判明することが多いが，ソフトウェアでの排他資源の競合は OS の稼働情報では分からないので，判明までに時間がかかることが多い．

競合の具体的な要因としては，以下のものがある．

・CPU
・メモリ
・DISK
・通信
・排他資源

(3) キャッシュによる性能トラブルについて

キャッシュに起因する性能トラブルは，原因究明に時間がかかることが多い．例えば，DISK の I/O では，ユーザープログラムがレコードをブロッキングしたり，OS のメモリキャッシュでの I/O，VM のキャッシュ，ハードウェアのキャッシュといろいろな階層でのキャッシュが存在している．つまり，それぞれの階層でキャッシュのサイズ，動作タイミング，動作特性などの相性があり複雑な動きとなる．これに I/O の同期，非同期の組み合わせも加わる．

例えば，ユーザープログラムでレコードをブロッキングし，ブロック単位で同期 I/O を実施した時に，DISK のキャッシュサイズより小さい時は，キャッシュに書き込んだ直後に戻るが，大きなサイズになると，実際の DSIK の円盤への書き込み処理との同期が発生して，非常に時間がかかるようになる．

また，実行時間の短いユーザープログラムで大量データの I/O を非同期で実行した場合，途中の I/O の時間は短いがファイルのクローズまたはプロセスの終了時に OS のキャッシュ上のデータの書き込み処理の同期待ちが発生し，プロセスの終了までに時間がかかることがある．

キャッシュの具体的な性能劣化要因としては，以下のものがある．

・ヒット率低下

例えば，DISK 上に A データ，B データ，C データ．．．Z データのように複数のレコードを連続して書き込んだファイルに対して，Z データから前の方向に読み込み処理をするとキャッシュのヒット率が低下することがある．DISK の読み込み処理の場合には，OS レベルなどで先読みしている場合が多く，順方向であれば，キャッシュに先読みしたデータがありヒット率が高くなるが逆読みすると先読みの効果が得られずにヒット率の低下となる．

・キャッシュの容量不足
・ハードウェアなどのキャッシュのアクセススピードが遅い
・キャッシュが無効

(4) ループによる性能トラブルについて

ループに起因する性能トラブルは，原因究明が短い時と長期化する時の2極化する傾向がある．ループする範囲が小さい場合には，原因究明が短いことが多いが，広い範囲でのループになるとループしてしまう原因究明に手こずることがある．特にループの中にネストしてループの処理を追加した場合には，ループの回数は積数となるので，どちらかのループ回数が増加しただけで急激に増加してしまう．

ループが起因するものとしては，以下のものがある．

・無限ループ
・非常に回数の多いループ
・リトライ回数が多くなる可能性があるループ
・ループ中のループ
・ループの処理中にシステムコールなどの重い関数を追加

(5) ムダな処理による性能トラブルについて

ムダな処理に起因する性能トラブルは，いろいろな原因が考えられるので究明に時間がかかることが多い．特に，想定していた処理件数では全く性能的に問題なく動作していたものが，件数の増加とともに指数関数的に性能が劣化するケースがあり，原因究明に時間がかかることがある．

ムダな処理の具体的な要因としては，以下のものがある．

・輻輳
・メモリ確保関数を頻発

・I/O 処理を頻発
・同期処理
・メモリのコピーの実装が悪い
・ソート, コンペアの実装が悪い
・サーチ, インデックスの実装が悪い
・キューイング処理の実装が悪い
・ループ中のループ

⑹ ムラな処理による性能トラブルについて

ムラに起因する性能トラブルは, 動作が特定することが難しく, 原因究明が長期化することが多い. 例えば, プロセスのプライオリティが低いような場合には, 他のすべてのプロセスがどのようにCPUなどを使用しているかで振舞いが変わるため, 該当プロセスだけの振舞いを調査しても原因究明には至らない. DISK の断片化による性能トラブルは, 突然発生するのではなく, 少しずつ性能低下していくので, 気づかないことが多い.

ムラな動作の具体的な要因としては, 以下のものがある.

・プライオリティ
・DISK の断片化
・メモリの断片化
・バッファサイズの不適切な値

⑺ その他の性能トラブルについて

性能トラブルは, ⑴から⑹のような事象に分類されることが多いが, プログラム固有の実装ロジックがあったり, ハードウェアの特性に起因することもある. 例えば CPU などでは, 動作時の温度によってクロック数が変わるものがある. これなどは, ソフトウェアを開発する人にとっては, 再現テスト時の観点として, ハードウェア特性の観点はなかなか想像できないことが多い.

その他の具体的な要因としては, 以下のものがある.

・同居製品固有の実装ロジック
・OS 固有の実装ロジック
・ライブラリ固有の実装ロジック
　例えば, 文字列を処理する append()関数などは注意が必要である.
・自プロセスの固有の実装ロジック
・通信品質の特性(特に無線 LAN)
・ハードウェア特性

4. 3 どのような現象が起きているかを具体的に調査する方法について

4. 3. 1 該当プロセスの CPU 使用率, メモリ使用量, DISK I/O 量, 通信量を確認する.

⑴該当プロセスの CPU 使用率が 0%の時

⑻CPU 使用率が 0%の時は, WAIT で止まっている可能性が高いので, 止まっている場所を特定する.
・プログラムのトレースログを確認して場所を推定する.
・スナップダンプを取得し, 場所を特定する.
・システムコールトレースから, 場所を推定する.
・システムコールであれば, システムコール名称とシステムコール時の引数を確認する.

⑹止まっている原因を特定する.
・排他処理であれば, 排他資源名を確認して, デッドロックになっていないかを確認する.
・タイマー設定後の WAIT 関数であれば, タイマー値を確認し, 値が大きすぎないかを確認する.
・システムコール等に続く WAIT 関数であれば, エラー後のリトライ間隔が大きすぎないかを確認する.
・システムコール等に続く WAIT 関数であれば, エラー後のリトライ回数が大きすぎないかを確認する.
・他プロセスと通信しながらの処理があり, 他プロセスからの返信待ちになっていないかを確認する.
・システムコール等で無限待ちの引数でコールしていないかを確認する.
・排他の範囲が大きく他のプロセスとの競合が発生し易くなっていないかを確認する.
・排他の範囲を小さくするために, 排他を細切れにして発行回数が多くなっていないかを確認する.

⑵該当プロセスの CPU 使用率が 100%に近い時

該当プロセスの CPU 使用率が 100%であっても, 8 コアの CPU だと, システム全体の CPU 使用率が 12. 5%と表示されるので, プロセス毎の CPU 使用率を確認する必要がある.

⑻プロセスの CPU 使用率が 100%に近い時は, ループしている可能性が高いので, ループしている場所を特定する.
・プログラムのトレースログを確認してループしている場所を推定する.
・スナップダンプを複数回取得し, ループしている場所を特定する.
・システムコールトレースから, ループしている場所を推定する.

⑹ループしている原因を特定する.
・スナップダンプからループカウンターの値を確認する.
・ループ箇所のソースコードから無限ループに陥ることがないかを確認する.
・ループ箇所のエラー処理で無限にリトライすることがないかを確認する.
・ループ箇所の中に排他エラーの時に無限にリトライすることがないかを確認する.
・ループの中にループするような処理がないかを確認する.

⑹実装方法の確認
・コピーの実装が悪い
・ソート, コンペアの実装が悪い
・サーチ, インデックスの実装が悪い
・キューなどの実装が悪い
・文字列操作などの実装が悪い

⑶該当プロセスのメモリサイズだけでなく, メモリの確保/解放回数も確認する.

⑻処理するデータ件数が多くなった時の振舞いを確認する.
・メモリの確保/解放のロジックを確認し, 件数に比例する以上のメモリを使用しないかを確認する.
・件数の増加に伴って, メモリの確保回数も多くならないかを確認する.
・レコード1件読む毎にメモリを確保し, チェーンでつないでいないかを確認する.

⑹処理するデータサイズが大きくなった時の振舞いを確認する.

・最初のデータ格納サイズが小さく，大きなデータになると少しずつメモリ増分していないかを確認する．
・C++のライブラリで「strcopy」などの関数を使用していないかを確認する．

(4)該当プロセスの DISK I/O の確認
(a)該当プロセスの DISK I/O 回数が非常に多い
コンピュータのパフォーマンスを測定するツールで DISK の読み込み回数，書き込み回数を確認する．
(b)該当プロセスの DISK I/O のサイズが大きい
コンピュータのパフォーマンスを測定するツールで DISK の読み込みサイズ，書き込みサイズを確認する．
(c)DISK の I/O を同期 I/O が不必要な所で実施していないかを確認する．

(5)該当プロセスの通信の量が多い時
(a)ネットワークのパフォーマンスを測定するツールで送信量，受信量を確認する．
(b)プログラムを確認して，プロトコルデータのサイズと通信バッファのサイズを確認する．
(c)OS の通信バッファのサイズを確認する．

4. 3. 2 他のプロセスの CPU 使用率，メモリ使用量，DISK I/O 量，通信量を確認する．
(1)他のプロセスの CPU 使用率が 100%に近い時
・プログラムのトレースログを確認して，他プロセスを呼び出しているかを確認する．
・プログラムのトレースログを確認して，他プロセスを頻繁に呼び出しているかを確認する．
・スナップダンプを取得し，他プロセスを呼び出しているかを確認する．

(2)system の CPU 使用率が 100%に近い時
・システムコールトレースから，システムコールが頻繁に発行されていないかを確認する．
・プログラムのトレースログを確認して，システムコールの呼び出しの延長かを確認する．
・プログラムのトレースログを確認して，システムコールの呼び出しを頻繁に実施しているかを確認する．

(3)全体の CPU 使用率が 100%に近いが，自プロセスの CPU 使用率が低い時
・自プロセスに十分な CPU が割り当てられていない状況かを確認する．
・自プロセスのプライオリティを確認する．

(4)他のプロセスのメモリの使用量が多い時
・システム全体のメモリ使用量を確認し，システム全体でページフォルトが頻発しているかを確認する．
・システムコールトレースで，システム全体のメモリの確保/解放が頻発しているかを確認する．
・スラッシングが発生しているかを確認する．

(5)システム全体の DISK I/O を確認
(a)システム全体の DISK I/O 回数が非常に多い
・DISK I/O のパフォーマンスを測定するツールで読み込み回数，書き込み回数を確認する．
(b)システム全体の DISK I/O のサイズが大きい
・DISK I/O のパフォーマンスを測定するツールで読み込みサイズ，書き込みサイズを確認する．

(6)システム全体の通信量が多い時
(a)送受信の量が多い
・ネットワークのパフォーマンスを測定するツールで送信サイズ，受信サイズを確認する．

4. 3. 3 通信相手の他プロセスを確認する．
現象が表面化するプロセスに問題がなくても，通信相手のプロセスからのレスポンスが悪いことがある．このような場合には，通信相手のプロセスの調査が必要になる．

(1)通信相手の他プロセスも自作のプログラムの場合
・通信相手のプロセスのトレースログなどを確認する．
・通信相手のプロセスの CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を調査する．
・通信相手のプロセスが他ホストの場合，他ホストのシステム全体の CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を確認する．

(2) 通信相手の他プロセスが自作のプログラムでない場合
・通信相手のプロセスの CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を調査する．
・通信相手のプロセスが他ホストの場合，他ホストのシステム全体の CPU 使用率，メモリ使用量，DISK I/O，通信などの負荷状況を確認する．

(3)自プロセスと通信相手のプロセスの間で通信の輻輳が発生していないかを確認する．

4. 3. 4 プロセスのプライオリティを確認する．
(1)プロセスのプライオリティの確認
・自プロセスだけでなく，他のプロセスも nice 値とプライオリティを確認する．

4. 3. 5 各種キャッシュの状況を確認する．
キャッシュは，ヒット率，キャッシュサイズ，キャッシュの種類，キャッシュの有効/無効などを確認する．

(1)DISK I/O 時の自プログラムのメモリ上のキャッシュを確認する．
例えば，複数レコードを1ブロックで書き込むような場合．
(2)DISK I/O 時の OS のメモリ上のキャッシュを確認する．
(3)DISK I/O 時の DISK 上のキャッシュを確認する．
(4)通信時の自プログラムのメモリ上のキャッシュを確認する．
(5)通信時の OS のメモリ上のキャッシュを確認する．
(6)通信時の通信装置によるキャッシュを確認する．
(7)メモリのキャッシュを確認する．
科学計算のようなインコアで処理するロジックがある場合には，
・CPU にある一時キャッシュのサイズ，二次キャッシュのサイズとアクセススピードを確認する．
・マシンが SMP(Symmetric Multiprocessing)構成かを確認する．VM の場合に，割り当てられるメモリキャッシュが他の CPU になる場合があり，メモリアクセス性能が悪くなる場合がある．

5. What(何を)は，原因となる資源を究明

性能トラブルが発生した時には，What(何を)に相当するソフトウェアが使用する資源を特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，What(何を)に相当する資源の観点で分類すると大きく6種類になる．図3は，性能トラブルの事象の原因をWhat(何を)の観点で分類したものである．

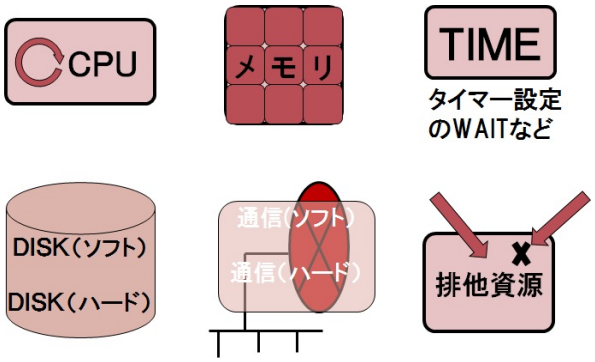


図 3. What(何を)の原因となる資源を究明時の観点

5. 1 性能トラブルがどのような資源で発生しているかの観点

性能トラブルが発生しているということは，性能が思ったように出ていないということであり，要因として資源が不足あるいは資源が競合していることが考えられる．不足するかは元の資源の能力との見合いなので，ハードスペックも確認する必要がある．究明するには，いろいろな資源に関係する性能資料を採取して解析すると有効である．

性能トラブルが発生している時は，資源として大きく「CPU」，「メモリ」，「TIME」，「DISK」，「通信」，「排他資源」に分類できる．性能トラブル解析時には，この 6 つのどの資源が原因で発生しているかを特定する必要がある．

5. 2 原因となる資源の絞り込み方法について

「CPU」，「メモリ」，「DISK」と「通信」については，「ハードウェアの能力」を「システムが使用する量」が超えることはできない．このため，最初にハードウェアのスペックと OS の各種統計情報を確認する必要がある．「TIME」は，WAIT などによって，ある意味「時間」という資源を消費していると考えて「資源」の 1 つとした．「排他資源」も WAIT で待つが競合するものが時間ではないので別物として扱う．

5. 2. 1 ハードウェアスペックを確認する．

(1) CPU を確認する．

・CPU の種類，CPU 数，コア数，クロック性能と CPU の 1 次/2 次のキャシュサイズを確認する．

・高温環境下だとクロック性能が抑制される場合があるので，動作温度を確認する．

・VM の場合には同一筐体内の他の OS の CPU 使用率も確認し，該当 VM に割り当てられているコア数を確認する．

・SMP(Symmetric Multi Processor)構成のマシンかを確認する．4 台のサーバを 1 個の OS で 1 個のマシンのように扱うため，OS 自身の排他処理の時はすべての他のマシンも含めて止まってしまうので4倍の性能にはならないことが多い．

(2)メモリを確認する．

・メモリの種類，メモリのサイズを確認する．

・SMP(Symmetric Multi Processor)構成のマシンかを確認する．4 台のサーバを 1 個の OS で 1 個のマシンのように扱うため，他のマシンの CPU のメモリキャッシュまで同期が必要になるので効率が悪くなることがある．

(3)DISK を確認する．

・DISK 単体の記録密度，回転速度，転送速度を確認する．

・DISK のキャッシュサイズを確認する．

・DISK のキャッシュのヒット率を確認する．

・DISK の接続方法を確認する．
SCSI，SAS，DAS，ファイバチャネル等．

・NAS であるかを確認する．

・DISK 装置とのケーブルの転送速度を確認する．

・DISK 装置の構成を確認する．
例えば，接続形態がデイジーチェイン(daisy chain)方式かを確認する．

・RAID の種類を確認する．

・RAID のキャッシュサイズを確認する．

・ディザスタ構成と方式を確認する．
実現方式がハードウェアかソフトウェアかを確認する．また，同期か非同期かも確認する．

・VM の場合には複数の VM から 1 つの DISK の玉にアクセスする構成の場合には，他 VM の DISK I/O の状況も確認する．

(4) 通信を確認する．

・論理的な通信速度を確認する．

・通信の帯域を確認する．

・通信相手との距離を確認する．

・VM で同一筐体内の通信かを確認する．

・NIC の情報を確認する．

・接続ケーブルの種類を確認する．
10BASE，100BASE，1000BASE など

・無線の場合は，周波数を確認する．

・途中のルーターなどで「SYN ACK」をシミュレートする装置かを確認する．

・ネットワーク構成図を確認する．

(5)画面表示に関係するものを確認する．

・グラフィックボードや GPU を確認する．

(6)プリンタを確認する．

・印刷がある場合には，プリンタのスペックを確認する．

(7)その他

・上記以外の入出力がある場合には，ハードウェアのスペックを確認する．

5. 2. 2 OS の性能情報から状況を確認する．

(1) OS の CPU 情報を確認する．

・システム全体の CPU 使用率とプロセス毎の CPU 使用率を確認する．

・各プロセスのプライオリティを確認する．

・VM の場合には同一筐体内の他の OS の CPU 使用率も確認する．
VM の設定によっては，VM の CPU 数が固定ではなく，最大 CPU 数と最少 CPU 数のような形式で指定できるものがある．

この場合には，他のVMが使用しているCPU量に応じてダイナミックにCPU数が変化するので注意が必要である．

(2) OSのメモリの情報を確認する．
- システム全体のメモリ使用量を確認する．
- プロセス毎のメモリ使用量を確認する．
  時間の経過とともに増減がある場合には，傾向も確認する．
- システム全体のページング状況を確認する．
- プロセス毎のページフォルト状況を確認する．
- 共用メモリの状況を確認する．
- 仮想メモリのサイズを確認する．

(3) OSのDISK情報を確認する．
- プロセス毎のDISK I/Oの読み込み回数とバイト数と書き込み回数とバイト数を確認する．
- OSのDISK I/OのI/Oチェーン数を確認する．
- DISK I/Oのキャッシュのヒット率を確認する．
- DISKがビジーな状態かを確認する．
- VMの場合には，他のVMマシンのDISK情報も確認する．

(4) OSの通信情報を確認する．
- OS全体の通信速度を確認する．
- NICやワイヤレスネットワーク毎の通信量を確認する．
- レスポンス時間を確認する．
- OSのバッファサイズを確認する．
- ポートの使用状況を確認する．
- 実際に最大でどのくらいの通信速度までででるかを確認する．
- ネットワークトレースを採取し，内容を確認する．

(5) 時間(=WAIT)の確認をする．
- WAITにより，長時間止まっていないかを確認する．プロセスのCPU使用率が0に近い値になっている．
  スナップダンプなどを取得し，WAITで止まっているかを確認し，止まっている理由がタイマー，排他資源，I/O，OS関数なのかを確認する．タイマーの場合には，WAIT時間を確認する．
- 短い時間のWAITが頻発しているかをログ等から確認する．ログがなくCPU使用率が予想より低ければ，スナップダンプなどを複数回取得し，WAITしている場所が毎回同じで止まっているかを確認し，止まっている理由がタイマー，排他資源，I/O，OS関数なのかを確認する．同じ場所の場合には，WAITが頻発している可能性がある．WAITの理由がタイマーの場合には，WAIT時間と頻発する原因を確認する．

(6)排他資源を確認する．
- 長時間WAITにより，長時間止まっていないかを確認する．プロセスのCPU使用率が0か，0に近い値の場合には，デッドロックの可能性がある．デッドロックは，2つのプロセスだけでなく3つ以上のプロセスの場合もあり，同じ排他資源を使用するプロセスの洗い出しが必要である．
  スナップダンプなどを取得し，WAITで止まっているかを確認し，止まっている理由が排他資源かを確認する．デッドロックであれば，同様の現象が発生している他のプロセスを探し，排他待ちであるかを確認する．
- 上記と同様の時に，資源の確保と解放で，排他資源名を間違えていないかを確認する．
- 短い時間のWAITが頻発しているかをログ等から確認する．ログがなくCPU使用率が予想より低ければ，スナップダンプなどを複数回取得し，WAITしている場所が毎回同じで，WAITで止まっているかを確認し，止まっている理由が排他資源かを確認する．同じ場所の場合には，WAITが頻発している可能性がある．排他資源が他のプロセスと競合していないかを確認する．
- 排他資源に対して，必要以上に頻繁に排他したり，排他範囲が広すぎないかを確認する．

6. When(いつ)は，原因となるきっかけを究明

性能トラブルが発生した時には，When(いつ)に相当するソフトウェアが性能トラブルを発生するきっかけを特定する必要がある．性能トラブルの事象の原因は多種多様ではあるが，When(いつ)に相当するきっかけの観点で分類すると大きく5種類になる．図4は，性能トラブルの事象の原因をWhen(いつ)の観点で分類したものである．

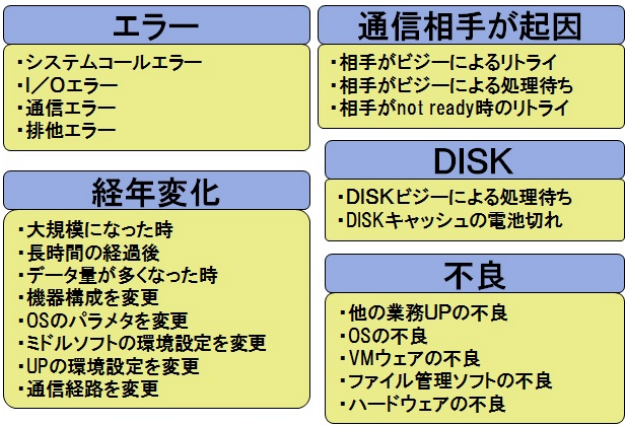


図4. When(いつ)の原因となるきっかけの究明時の観点

6. 1 性能トラブルが発生したきっかけの観点

性能トラブルの発生は，今までは問題なく動作していたが，あるきっかけを契機に発生することがある．

昨日までは，問題なく動作していたものが，なぜか突然に性能が悪くなることは，要因としてなんらかの変化があったと考えられる．昨日と何が変わったのかは，いろいろなログ，操作履歴や構成変更履歴に関係する資料を集めて解析すると有効である．

性能トラブルが発生するきっかけとしては，大きく「エラー」，「通信相手が起因」，「経年変化」，「不良」，「DISK」に分類できる．性能トラブル解析時には，この5つのきっかけが原因で発生しているかを特定する必要がある．

6. 2 原因となるきっかけの絞り込み方法について

(1) エラーを確認する．

エラーの発生の有無は，性能トラブルが発生した製品のログだけでなく，システムのログも確認する．エラーの発生がいつからか，何回発生しているかを確認し，性能トラブルとの関係の有無を確認する．場合によってはハードウェアのログも確認する．エラーの種類としては，大きく以下の4種類になる．
- システムコールエラー
- I／Oエラー

・通信エラー
・排他エラー

(2) 通信相手が起因しているかを確認する.

他のマシンと通信があるプログラムの場合, 性能トラブルは製品が動いているマシンだけを調査しても判明しないことがある. 通信相手や通信の状況に起因して性能トラブルが発生することがある. 次の 3 つの観点で調査して, 通信相手が起因しているかを追究し, 場合によっては通信相手側のプログラムの調査を実施する.

・相手がビジーによるリトライ
・相手がビジーによる処理待ち
・相手が not ready 時のリトライ

(3) 経年変化の要因を確認する.

性能トラブルは, 突然発生する場合もあるが, 時間の経過とともに少しずつ遅くなることもある. 少しずつ遅くなる場合には, 気づくのが遅くなることが多く, 原因究明も難しくなることが多い. これは, 長い時間の間には変化する対象が多くなるため, 原因究明が難しくなる.

・大規模になった時
・長時間の経過後
・データ量が多くなった時
・機器構成の変更
・OS のパラメタを変更
・ミドルソフトの環境設定の変更
・UP の環境設定の変更
・周辺装置の変更
・通信経路の変更

例えば, LAN と無線の両方を使用している場合に, LAN が有効な状態でもある時から無線が使われることがある.

・DISK のフラグメンテーション
・メモリのフラグメンテーション
・ハードウェアのファームウェアの変更

(4) DISK の状況を確認する.

最近は, 複数の DISK が複数のコンピュータと接続されているケースが増えている. このため, 1 つの DISK が複数のコンピュータからもアクセスされることがあり, 他のコンピュータの I/O が頻繁にあると影響を受けることがある.

・DISK の接続構成の確認
・DISK ビジーによる処理待ち
・DISK キャッシュのヒット率
・DISK キャッシュの電池切れ

(5) 各種不良がないかを確認する.

性能トラブル発生時でも, OS やハードウェアなどの既知の不良がないかを確認する.

・他の業務UPの不良
・OS の不良
・VM ウェアの不良
・ファイル管理ソフトの不良
・ハードウェアの不良

7. Whom(誰によって)は, 原因を誘発したものを究明

性能トラブルが発生した時には, 自分のプログラムに関してだけ調査しても判明しないことがある. 時々ではあるが, 自分ではなく他からの影響を受けている場合があり, Whom(誰によって)を特定する必要がある. 性能トラブルの事象の原因は多種多様ではあるが, Whom(誰によって)から誘発されたかの観点で分類すると大きく3種類になる. 図5は, 性能トラブルの事象の原因を Whom(誰によって)の観点で分類したものである.

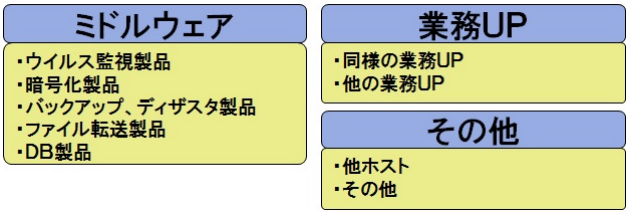


図 5. Whom(誰によって)の原因を誘発したものの究明時の観点

7. 1 性能トラブルの原因を誘発しているものの観点

自分のプログラムの性能が悪いとどうしても, 自分のプログラムに問題があり, その原因究明に注力してしまい, 問題が見つからずに長期化することがある.

このため, 性能トラブル発生時には, 問題が発生している自分のプログラムだけでなく, システム全体の性能情報の解析から始めることが大切である.

性能トラブルを誘発する原因としては, 大きく「ミドルウェア」, 「業務 UP」, 「その他」に分類できる. 性能トラブル解析時には, この 3 つの他の誰かによって誘発されて発生しているかを特定する必要がある.

7. 2 原因となる他の誰によって誘発されているのかの絞り込み方法について

(1) ミドルウェアを確認する.

同一マシンにミドルウェア製品がインストールされている場合には, ミドルウェアが大量のリソース(CPU, メモリ, DISK I/O, 通信)を消費している場合があり, その影響を受けて性能トラブルが発生している場合がある.

・ウイルス監視製品

UP の I/O や通信の延長で動作する場合があり, パターンファイルの更新を契機に性能遅延が発生することがある.

・暗号化製品

DISK I/O の延長で書き込み時の暗号化と読み込み時の複合化処理が動作することがあり, データサイズが大きいと CPU を消費することがある.

・バックアップ製品

バックアップ製品実行中は, 高速化のためにデータサイズを非常に大きくして書き込むことがある. このため RAID のキャッシュなどを大量に消費し, UP の I/O のキャッシュのヒット率が低下するなどの影響を受けることがある.

・ディザスタ製品

ディザスタ製品は, 同期/非同期, ハードウェアによる実装だけでなく, ソフトウェアによって実装されるものもある. 特に, 同期の実装方法によっては, 性能に大きく影響する場合がある.

・ファイル転送製品

ファイル転送製品実行中は，通信がビジーになる可能性が高くなる．製品によっては複数のコネクションを接続して同時に通信する製品もあり，UP が使用できる帯域が小さくなり，通信の待ちキューも長くなる場合がある．

・DB製品

DB 製品実行中は，大量の I/O だけでなく，同時に大量に CPU やメモリを消費することがある．複雑な SQL などを並行で実行すると，インコア処理が並行して実行するので，複数の CPU 資源を大量に消費することがある．

(2) 業務 UP を確認する．

同一マシンには，いろいろな業務 UP が動作しており，他の業務 UP が大量のリソース(CPU，メモリ，DISK I/O，通信)を消費している場合があり，その影響を受けて性能トラブルが発生している場合がある．

・類似の業務 UP

業務 UP の中で似ている UP は，消費するリソースも似ている傾向があり，同時に多くの類似した UP を実行すると，性能トラブルが発生することがある．特に，排他資源で同一の資源をシリアライズしている場合には，性能トラブルが発生し易いので注意が必要である．

・異なる業務 UP

1 つのシステムでも，実行される業務 UP は，いろいろとありかつ，多数の業務 UP が同時に実行されていることがある．いつもであれば，同時に実行されることがない UP でも，データ件数の変化などにより，同時に実行されてしまうこともある．そのような時に，I/O が集中してしまうことがあったり，同一 DISK の同一ドライブに負荷がかかることがある．性能トラブル発生時は，自分の UP だけでなく，同時期に実行している他の UP も調査する必要がある．

(3) その他．

・他ホスト

他ホストと連携して処理するような UP の場合には，他ホストからのレスポンスが悪いと，性能トラブルが発生することがある．

・ウイルスソフトに感染している

ウイルスソフトに感染するといろいろなファイルへのアクセスや通信が発生し，性能トラブルを誘発する．

・その他

Dos 攻撃や Ddos 攻撃を受けている場合には，OS などがその対応で CPU を消費し，UP が CPU を十分に使えなかったり，通信速度が低下するなど性能トラブルが発生することがある．

8. 性能トラブル発生時の具体的な調査方法

はじめにで記述したように，本論文を作成する前に現象などから調査ポイントを段階的に実施すれば原因究明できるような資料として，「性能トラブル解決の手引き−事例編」としてソフトウェア・メインテナンス研究会で作成したものがある．

123個の事例を表形式でまとめたものである．表には，項目として「現象」，「調査ポイント 1」，「確認結果 1」，「調査ポイント 2」，「確認結果 2」，「原因」，「調査場所大分類」，「調査場所中分類」，「調査場所小分類」，「観点」，「調査観点」，「備考」などがあり，現象から調査ポイントの確認結果の内容に応じて段階的に絞り込むことで，原因を究明するものである．表 1 は，「性能トラブル解決の手引き−事例編」の抜粋である．

「性能トラブル解決の手引き−事例編」の全文は，ソフトウェア・メインテナンス研究会の HP に掲載してある．

http://www.serc−j.jp/

また，ログなどの調査を開始すると同時にシステム変更点，運用の変更点なども同時に確認する．

(1)システム変更の確認

・ハードウェア構成の変更

・新たに導入したプログラムの有無の確認

・ソフトウェアのアップデートの確認

・ソフトウェアのパラメタ変更の変更

(2)運用の変更の確認

・いつもと違うオペーレションを実施していないかの確認

・週末・月末などの業務量などの変化の確認

	現象	調査ポイント1	確認結果1	調査ポイント2	確認結果2	原因	調査場所大分類	調査場所中分類	調査場所小分類	観点	調査観点	備考
55	処理が重い	メモリ使用量の確認	メモリ使用量が急激に増加しているかを確認する。	(1)メモリサイズの時間の経過による推移を確認する。 (2)スナップダンプによる解析(メモリの使用状況の確認) (3)ソースでメモリの確保、開放処理の洗い出し。 (4)システムコールトレースでメモリ確保関数の回数の確認	(1)メモリサイズの変化とシステムコールトレースから、大きなメモリサイズの確保が、小さなサイズを大量に確保したかを確認する。 (2)ダンプを解析し、確保したメモリサイズと数を確認する。 (3)ソースでメモリの確保時のサイズと回数を確認する。	(1)メモリの確保サイズが巨大。 または (2)メモリの確保する個数が膨大	ソース	メモリの確保/開放のロジック		メモリの確保/開放	時々だが急激に重くなるような状況があるかを確認。	OSによっては、大きなメモリを確保しても、実際にメモリへの書き込みがないと実メモリを割り当てないケースがあるので、大量のメモリへの書き込みを同時に実行しないと発生しないこともある。
56	処理が重い	メモリ使用量の確認	メモリ使用量が急激に増加しているかを確認する。	(1)メモリサイズの時間の経過による推移を確認する。 (2)スナップダンプによる解析(メモリの使用状況の確認) (3)ソースでメモリの確保、開放処理の洗い出し。 (4)システムコールトレースでメモリ確保関数の回数の確認	(1)メモリサイズの変化とメモリの確保/開放のシステムコールが大量に発行されていないかを確認する。 (2)ダンプを解析し、確保したメモリサイズと数を確認する。 (3)ソースでメモリの確保時のサイズと回数を確認する。	データコピー時にコピー先の領域を1バイトずつ拡張しながらコピーしている。 データサイズが大きくなると指数関数的にメモリ処理量する。またCPUも消費し処理に時間がかかる。	ソース	メモリの確保/開放のロジック		メモリの確保/開放	メモリ確保のシステムコールが大量に発行されていないかでわかる	speedが関数などだと関数の延長でこのような実装がされているので注意が必要。
57	処理が重い (長時間経過後に時間の経過と共に遅くなる)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	時間の経過と共にCPU使用率が増加しているかを確認する。	(1)スナップダンプによる解析(メモリの使用状況の確認) (2)ソースでメモリの確保、開放処理の洗い出し。 (3)システムコールトレースでメモリ確保関数の回数の確認	(1)ダンプを解析し、メモリが断片化していないかを確認する。 (2)ソースでメモリの確保時のサイズより小さいサイズで開放するなどの処理がないかの確認。 (3)メモリの確保サイズが可変で頻繁に実行されていないかの確認。	メモリの断片化。 100kバイトを確保し90Kバイトだけ解放のような処理を繰り返すと、メモリが細片化してメモリの確保に時間がかかるようになる。)	ソース	メモリの確保/開放のロジック		メモリの確保/開放	・自分のソースのメモリの確保/解放処理 または、使用しているライブラリの延長で発生していることもあるので、注意が必要。	・メモリ使用量(見た目の使用メモリは少なくてもフラグメンテーションのためメモリ不足エラーになることがある)
58	処理が重い 但し一時的 (長時間経過後に時々急激に遅くなることがある)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	CPU使用率の時間の経過による推移を確認する。	時々CPU使用率が高い状態が発生し、やがて解消されるような傾向があるかを確認する。	(1)一時的に重い時に、CPU使用率が以上に高いかを確認する。 (2)OSがJavaのようなメモリのガーベージコレクションが実行していないかを確認する。 (3)ダンプ解析でメモリの内容を確認する。	OSのガベージコレクションが実行されている。	OS	ガベージコレクション処理の実装方法		ガベージコレクション処理	時々だが急激に重くなるような状況があるかを確認。	
59	処理が重い 但し一時的 (長時間経過後に時々急激に遅くなることがある)	CPU使用率の確認 UNIX,Linuxの場合には、sysとusrの比率も確認	CPU使用率の時間の経過による推移を確認する。	時々CPU使用率が高い状態が発生し、やがて解消されるような傾向があるかを確認する。	(1)一時的に重い時に、CPU使用率が以上に高いかを確認する。 (2)Javaのメモリのガーベージコレクションが実行していないかを確認する。 (3)ダンプ解析でメモリの内容を確認する。	Javaのガベージコレクションが実行されている。	Java	ガベージコレクション処理の実装方法		ガベージコレクション処理	時々だが急激に重くなるような状況があるかを確認。	使用しているJavaによって挙動が異なる。
60	処理が重い 突然遅くなる	CPU使用率の確認	CPU使用率が高い状態が続く	(1)システムコールトレースで通信の回数を確認する。 かつ (2)該当プロセスの通信量を計測する。 (3)ソースを解析し、通信処理の周辺のロジックを確認する。 かつ (4)通信相手(他ホストの場合もある)のプロセスも高負荷状態にあるかを確認する。	(1)通信の回数が非常に多いかを確認する。 (2)通信のデータ量が多いかを確認する。 (3)通信相手が受け取れない時に、再送処理があり 輻輳状態になるロジックがあるかを確認する。 (4)通信相手のプロセスがビジーな状態で通信を受け取れない状況であるかを確認する。	他プロセスとのデータの受け渡しがあるが、他プロセスが処理しきれないデータを戻す処理があり 2つのプロセス間の通信処理で輻輳が発生している。	ソース	通信処理のロジック		輻輳	件数とサイズを変化させて性能測定し、グラフ化する。 ・いろいろな件数で処理時間の計測	通信相手のプロセスが処理しきれている間は、問題が発生しない。受信プロセスの処理性能 < 送信プロセスの送信量 の状態が続くと限界を超えると急激に遅くなる。

表1. 「性能トラブル解決の手引き−事例編」の抜粋

9. 性能向上する時の観点

性能トラブルが発生した時にデッドロック等のような明らかな不良であれば，不良を修正すればよいのだが，レスポンス向上や大規模に対応するために性能向上したい時に，どのような手法があるかの観点毎に説明する．

性能向上には，ハードウェアによる対策，ソフトウェアによる対策とハードウェアとソフトウェアの両方の対策がある．

How(どのように)の観点で見直すことと，What(何を)の観点のリソースを上手に使用することに着眼して見直すとよい．場合によってはリソースを贅沢に使用するようにして，ハードウェアを増強するという方法もある．

最近は，ハードウェアの価格が安くなってきているので，CPU の増設やグレードアップ，メモリの増設，DSIK の増設やキャッシュの増設，通信の高速化，帯域の拡大などで性能向上が可能であれば，プログラムの修正コストより安い場合もあり，ミスの危険もなく確実である．

ソフトウェアの対策には，以下のものがある．

⑴多重度を上げる．
- ・マルチプロセス化．
- ・マルチスレッド化．
- ・マルチインスタンス化．
- ・並行処理化．

⑵分散して処理する．
- ・複数エージェントでの分散実行．
- ・ロードバランサーでの分散実行．

⑶非同期にして処理する．
- ・I/O などの非同期処理．
- ・自立実行．
- ・見た目と実体の非同期処理．
- ・先読み処理．

⑷独自にキャッシュを実装する．
- ・キャッシュにより実 I/O より書き込み時間を短くする．
- ・キャッシュにより実 I/O より読み込み時間を短くする．

⑸まとめて処理する．
- ・I/O 時のレコードのブロッキング化．
- ・I/O サイズを大きくする．
- ・要求をまとめて処理する．

⑹整理する．
- ・デフラグする．
- ・ガベージコレクションする．
- ・差分のみの更新する．
- ・データは，ソート処理する．

⑺階層化
- ・Manager と Agent の構成にする．
- ・中継サーバーを構成する．
- ・インデックスを作成する．

⑻事前に準備する．
- ・先読み処理をしておく．
- ・プロセスを常駐化する．
- ・実 I/O 時に先読みしてキャッシュに入れておく．
- ・まとめて読み込みを行う．

⑼排他処理の見直しをする．
- ・排他時間を短くする．
- ・排他をしないようにする．
- ・排他範囲を小さくする．
- ・排他回数を減らす．
- ・排他エラー時のリトライ間隔を短くする．

⑽タイマー処理を見直しする．
- ・タイマー時間を短くする．
- ・タイマーのかける回数を減らす．
- ・タイマーをかけなくする．(非同期にするなど)

⑾その他
- ・ストリーム処理で実現する．
- ・ハッシュを使用する．
- ・アセンブラ化する．
- ・I/O サイズをハードのスペックに合わせる．
- ・ハードウェアの特性に合わせた実装を行う．
- ・ウイルスチェック対象から外す．

10. このドキュメントの対象について

(1) プラットフォームは，Windows，Linux，UNIX 上で動作するプログラム

(2) 通常のユーザープログラムまたは，ミドルウェアソフトのプログラム

(3)database，Application Server，Web server などのチューニングは対象外

(4) DISK，通信装置，機器構成などのチューニングは対象外

(5) OS のチューニングは，対象外

(6) VM ウェアに関しては，一部だけ対象としている

11. おわりに

個人的な経験をもとに性能トラブル解決の勘所としてまとめたが，まだ網羅性が不十分な状態であるので，この資料をベースに各自で追記して使用して頂ければ幸いである．

謝辞

本研究の資料まとめの際にソフトウェア・メインテナンス研究会の皆様に多くの助言を受けた．ここに謝意を記す．

参考文献

[1] 増井和也， 弘中茂樹， 馬場辰男， 松永真 ″ソフトウェア保守開発:ISO14764 による″ソフト・リサーチ・センター， 2007

[2] 保田 勝通，奈良 隆正 ″ソフトウェア品質保証入門″日科技連出版社，2008

[3] 鈴木勝彦 ソフトウェア・メインナンス研究会 第 24 年次報告書 D グループ報告 個人研究「性能トラブル解決の手引き」

レビュー要求分析・設計・実装試行でわかったこと

安達 賢二
HBA
adachi@hba.co.jp

要旨

2009 年からこれまで、SQiP シンポジウムや客先などのさまざまな場面で、「レビューの問題点」を収集してきた。その結果、ソフトウェア開発・保守を中心とした実務において、レビューの問題点が多数存在し、手間がかかる割に効果が実感できないという意見が多いことがわかった。

その打開に向けて、要求仕様書を題材にしたレビューワークショップを構築・提案し、実施した結果、欠陥検出を目的としたレビューにおいて「アドホックレビュー」よりも「レビュー観点設定に基づくレビュー」の方が指摘件数、指摘内容の両面において有効性が高いという結果を得た。
一方で、同一ワークショップに取り組んだ各チームが設定した観点（大項目）とその詳細（具体的な個別確認項目）、および欠陥指摘件数、指摘内容には多くのバラつきがあり、その打開に向けた対策が必要である。

今回は、レビュー観点（大項目）とその詳細（具体的な個別確認項目）のバラつき防止・低減を目指して「レビュー要求分析・設計・実装」を試行し、その効果を確認した。

その結果、以下のことがわかった。
(1)レビュー観点、確認項目の抜け・漏れ防止への効果が期待できる。
(2)併せて、レビュー実施の効率化が期待できる。
- ・論理的な観点単位にバラバラに確認する必要がなくなる
- ・確認対象が小さく、確認項目が具体的なので確認と判定が楽にできる
- ・関連する項目を連携しながら系統だって確認できる
- ・重複している項目が集約できる
- ・対象毎に不必要な項目を省くことができる

(3)テスト要求分析（の多くの部分）とテスト設計の一部を兼ねることができる。
(4) レビュー対象成果物作成作業（今回の例では要求定義）の一部を兼ねることができる。
- ・これができると、レビュー時に確認する観点・項目をリスクが高いものに絞り込むことが可能となる

(5)一方で、レビュー要求分析～実装まで手間と時間がかかる。

(5)については、レビュー要求分析～実装までの実践力がつけば多くの部分で打開できると想定される。また、さらに効率化するためには「レビュー観点図」（例：添付スライド 15・16）などの運営基盤となる情報を育てていくなどの対策が必要である。

参考文献

[1] SS2015 WG6 Position Paper「レビューの会議術からの脱却」電気通信大学 西康晴
[2] JaSST'16 東京 発表事例「レビュー目的・観点設定の効果と課題」 安達賢二

ソフトウェア開発現場での Minute Paper の適用

岡本　克也
有限会社エヌ・エム・エス
okamoto@nms.ne.jp

中谷　多哉子
放送大学
tinakatani@ouj.ac.jp

黒須　正明
放送大学
kurosu@ouj.ac.jp

要旨

要求工程での打ち合わせにおいてコミュニュケーション不良とクライアントの参加意識の低さがしばしば見られる. これらを早めに検出して関係者の参加意識を高め, ベンダーとクライアントが製品に対して同じイメージを共有することが望ましい. その為にアクティブ・ラーニング手法のひとつである Minute Paper をソフトウェア開発に適用し, その結果を横断的に分析することでコミュニケーションの不良を軽減し, 参加意識を高めることでその有用性を確認した.

1. 序論

筆者は過去に経験した複数の小規模な開発現場で発生した関係者の満足度低下につながる問題の原因について分析を行ったところ，多くの事例で

1. コミュニケーション不良によってお互いの考えていることが一致しない
2. クライアントの参加意識が必ずしも高くない

という事象が見いだされた.

ここで参加意識を定義する. この参加とは開発工程の全体において製品の開発に関わる他者の考えや行動に変化を与えようとすることである. 参加意識とは参加を意図して個人の内で行われる考察や試みである. 通常, 発言や行動の内容・頻度として発露される.

参加意識が低いとは参加の意図に乏しく開発に対する理解や探索のための考察や試みに費やす時間が少なく, 発言や行動に現れない状態を指す. 参加意識が高いとはその逆で参加の意図を持ち考察や試みに時間を充て, 発言や行動として外部に現れる状態である.

関係者はゴールとなる完成像について同じイメージを共有したい，あるいは違うなら違うで合意形成を計りたいのだが，1. の自覚がないと勘違いは最終的な製品の段階まで持ち込まれることになる.

2. の場合，参加者に関心の濃淡があるのはやむを得ないことであるが，極端に参加意識が低いと発言そのものがされないことがあり，他の参加者も感心の薄いクライアントを薄いまま放置しがちになる. しかし，クライアントは自身の欲求を持っており，この欲求はリリース直前，あるいはリリース後に表面化する.

打ち合わせの現場では, 関係者全員が揃う会議が持たれる回数は限られており, 特に小規模な事案ではクライアント側の担当者が指定されるものの, 担当者はいままで通りの業務を抱えており「ベンダー側の仕事」である開発に時間を割きたがらない傾向もある. そのため空き時間にベンダーがお話を伺うべく「個別に」インタビューを面接, 電話, メールなどで行うことになり, するとクライアントの横方向のすり合わせがどのように行われているか把握しづらく, さらに, 個人的な関係の上で情報がやりとりされがちな傾向が生まれる.

それが会議室であれ, 一対一のインタビューであれ, コミュニケーション不良と参加意識の低下は起こっている. 目の前で話をして出た「それでいいですよ」という言葉がよく考えた結果なのか, インタビューを切り上げて自分の仕事にあるいは別の話に戻りたいということなのか明確にならないことも事例からは散見された.

しかし, ここがクライアントの要求を聞き, ベンダーが説明をし, お互いが何を理解しているのか確認する, 極端に言えばすべてが行われる“現場”に他ならない.

「打ち合わせ」はベンダーとクライアントの唯一の接点である.

その重要性を考えれば「打ち合わせをデザインする」ことでその価値を高めることが必要となる.

2. 方法

筆者は MOOC(Massive Open Online Course)サイトである gacco の「インタラクティブ・ティーチング[1]」(主任講師:栗田 佳代子 東京大学大学総合教育研究センター教育課程・方法開発部門特任准教授 ならびに中原 淳 東京大学大学総合教育研究センター教育課程・方法開発部門准教授)を受講した際にアクティブ・ラーニング手法について, ソフトウェア開発への応用が可能でないかと考えた.

教育分野で近年導入されはじめたアクティブ・ラーニングとは「能動的な学習」の総称である.

教師が一方的にしゃべり, 学生が静かに話を聞いてノートを取る. という受動的な学習では課題に対する理解や考察を深めるという目的に効果的ではない. また, 学生には授業に対する関心に濃淡があり, 特に関心の薄い学生を取り残すことはその学生に対する理解を放棄することにつながる.

アクティブ・ラーニングは学生を積極的に参加させるべくデザインされた授業の技法を含む.

ここで筆者はひとつの見立てを行い, 学習を要求獲得のための打ち合わせ, 課題をベンダーとクライアントが共有したい要求の形と置き換えてみた.

ベンダー側が単に質問をし, クライアントがそれに答えるという形式では望ましい成果はなかなか得られない. 要求獲得段階でベンダーもクライアントも何を作り上げようとしているのか理解すること・探索することを重視すべきである.

とはいえ, クライアントにお願いして開発に参加してもらう場合, 時間的制約は常に問題となることもあり, ソフトウェアに対する理解や考察を効率的に深めるために良くデザインされた打ち合わせ技法へのヒントをアクティブ・ラーニングから得られないか, 主要な技法からソフトウェア要求獲得への応用の可能性を考えた.

教育現場で用いられるアクティブ・ラーニング手法のいくつかをここで紹介したい.

Minute Paper は授業中に配布するペーパーに授業への興味, 関心, 疑問, 自分の理解度などを数分で書いて提出してもらう技法である. 事前に用意した質問紙を用いるので実施にかかる時間も少なく, 記入する側の負担も小さい. 授業規模の大小にも柔軟に対応できる. これは学生側にとっては授業に求めるもの, 疑問の行き先, 困難を抱えているかを教師側に伝える方法であり, 教師側にとっては授業が意図したように進んでいるかを確認し, 今後の授業の進行変更を考える際の資料となる.

Think-Pair-Share は, 明確な課題を設定し,「一人で考える」「ペアになる」「考えたことを共有・議論・意見交換する」という順序で行われる「意見交換の前に考える時間を確保する」という 議論への主体的参加を促す構造を持つ技法である.

最少人数は2人, 時間的にさほど長くならず, 事前準備も簡単である.

ジグソー法は, 理解したい課題をまず分割し参加者を課題数のグループ(エキスパートグループ)に分ける. エキスパートグループは課題について調査し理解を深める. 次に, 参加者をシャッフルしそれぞれのエキスパートを含む新グループ(ジグソーグループ)を作成し, エキスパートは自分が調査した分割課題について説明を行い, また他の分割課題についての説明を聞いて情報を共有する. 最後に元のグループに戻って全体課題についての共有を行う. この方法の特徴はジグソーグループを作成した際に「自分しか知らない」状態が強制的に作り出され, それをグループで共有することがデザインされている点である.「調査」した結果を効果的に共有できるが, ある程度の人数と時間が必要となる.

ポスター法は上記のジグソー法をポスター発表に応用したものである. ポスター制作をエキスパートグループが行った後, ジグソーグループを作成して各ポスターを巡回して討論する. 全員にプレゼンテーションの機会があり「ただ乗り」を許さないが, この手法は人数, 時間, 準備, 道具が必要となる.

ピア・インストラクション法は知識獲得型にメリットのある技法で, 予習教材を提示した後 Concept Test と呼ばれる多肢選択問題を用意し参加者に実施する. 回答率を高中低に分け, 高い場合は次の Concept Test へ, 中ぐらいの場合はペア・小グループで議論を行い互いに教えあう. 低い場合は教師が再度説明を行う. 事前に全員に周知しておいて欲しい知識があるときに適しているが質の高い Concept Test を準備するには労力が必要となる.

今回, 中小企業の伝票管理業務の案件に, アクティブ・ラーニング手法から Minute Paper の技法を適用することを試みた.

この手法を選定した理由は, ひとつには参加者に対する時間的な負担が少ないことが挙げられる. 序論に述べたように小規模な開発の場合, 関係者が揃った開発会議というのは数回行えれば良い方で, 多くの場合, 業務を行っているユーザー側の時間の隙間を借りるという形で開発者側が打ち合わせに訪問したりすることになる. このような状況では参加者の時間的な負担を小さくすることは重要な要素になる. 今回は記入時間として5分＋程度を想定した.

本来であれば議事録を明確にしておくべきであり, それによってトラブルのいくつかは解消されると考えられるが, アクティブ・ラーニングの技法に接して"能動的"にインタラクション(教育現場では授業であり, 開発現場では打ち合わせである)をデザインするという視点に気づかされた. この点で従来の議事録とは異なるコミュニケーション改良への効果が期待された.

よってふたつ目の(重要な)理由として打ち合わせ後

に”振り返る”ことによって，なにを目的にした打ち合わせだったのか，ちゃんと意見交換は出来たのか，取りこぼしはないか等を考える時間を参加者に持ってもらうようにデザインしたことが挙げられる．この考える時間は重要であり，打ち合わせが終わったら速やかに関心が消えてしまうことを防止する効果が期待できる．

また，Minute Paper を書くことそれ自体が，開発に対する参加意識の向上に役立つことを期待して本研究ではこの手法を使用することにした．

今回，この手法を伝票管理アプリの開発の要求獲得段階で適用した．既存の伝票管理アプリのリプレース案件でありソフトウェアの規模も小さい．

また，利用者は経理担当者一人であり開発者も筆者一人という最小限の環境となっている．

これらことを考慮した上で，次のような項目を設定した．（付録参照）

1. 本日の打ち合わせで最も重要だと感じたこと，印象に残ったことは何ですか？
2. 言いたいと思ったことはいえましたか？言いそびれたことがあればお書きください．
3. 相手の話は理解できましたか？　分からなかったことがあればお書きください．
4. 相手はあなたの話を十分理解してくれたと思いますか，まだ理解が不十分かもしれないと思われるところはありませんでしたか？
5. 次回の打ち合わせで改善して欲しい点，期待することはありますか？
6. 打ち合わせ後に思いついたアイデアなどがあればお書きください．(文または絵)

この Minute Paper を打ち合わせ当日の終業前か，翌日に記入してもらうように依頼する．

配布対象はクライアント・ベンダー双方の関係者全員である

回収後，一回の打ち合わせに対して，あるいは同時期の参加者全員分（今回の例は2名）を横に並べて検討する．スプレッドシートなどにまとめても良い．

3. 結果

Minute Paper 配布の2日後，次のような回答を得た．

1.本日の打ち合わせで最も重要だと感じたこと，印象に残ったことは何ですか？
[クライアントの回答] 必要な時に必要なデータが即時閲覧できる，使いやすいソフトの開発をお願いします．
[ベンダーの回答] 良いものを早くできるように求めてると感じた

2. 言いたいと思ったことはいえましたか？言いそびれたことがあればお書きください．
[クライアントの回答] 今のところありません，入力ができるようになれば色々と出てくると思いますので
[ベンダーの回答] なし

3. 相手の話は理解できましたか？　分からなかったことがあればお書きください．
[クライアントの回答] 大変良く理解できました，分からない所はありませんでした．
[ベンダーの回答] 伝票の税込みの解釈で少し相違があるようだった．わかってくれたようだが，最初どう思っていたのかもう少し確認すべきかも

4. 相手はあなたの話を十分理解してくれたと思いますか，まだ理解が不十分かもしれないと思われるところはありませんでしたか？
[クライアントの回答] 十分理解してくれたと思います．
[ベンダーの回答] 理解してくれたと思う．

5. 次回の打ち合わせで改善して欲しい点，期待することはありますか？
[クライアントの回答] 今回の打ち合わせで，ほとんどご理解頂けたと思いますので，次回は是非画面イメージ等を見せて頂き動作検証ができるところまで行けばありがたいです，よろしくお願いいたします．
[ベンダーの回答] なし

6. 打ち合わせ後に思いついたアイデアなどがあればお書きください．(文または絵)

[クライアントの回答] (11月－翌年10末)等，期間を決めて未回収リスト等を作成できるようにして頂けると有難いです． 思い付いた事が出てきましたらご連絡致しますのでよろしくお願いいたします．
[ベンダーの回答] なし

4. 考察

4-1. 回答から得られたものを分析する

1. によると，クライアント・ベンダーはそれぞれ違うことを印象的に考えている．

クライアントにとっては打ち合わせでは，使いやすく，目的とする情報にすぐにたどりつけるソフトウェアについて要望したと認識している．

一方，ベンダーは Minute Paper に関して尋ねたときの「早くできるなら」という部分に反応している，何かするのであれば良い品質につながるのと同じくらい，仕上がりの早さを期待されていると認識している．締め切りを設定されるベンダーの習慣とも言えるが 5. のクライアント側の回答からそれが間違った印象では無いことが読み取れる．

2.は打ち合わせ中に言えなかったことを拾い上げる項目である．今回は双方ともに必要なことは発言できたと考えている．

3.4.は，それぞれ相手が自分の話を理解しているかと思うか尋ねている．本当にすべて理解されているかはともかく，「話はしたけれど分かっているのか」という疑念ポイントが生じた場合，早めに解消しておく必要がある．今回の回答ではベンダー側は説明を分かって貰ったと思っているものの，最初の相違はどのようなイメージの違いで生じたものか気にしている．この点についてベンダー側は翌日，電話にて再確認を行った．

5.は次回打ち合わせの改善点を尋ねている．実際は次回の打ち合わせではこんなことを話したいという議題の予告になっている．

ここでクライアントは次回では動作するものを見せて貰い検証ができることを期待している．行いたいことは 2. の入力ができて操作周りを確認することだと思われる．

ベンダー側は特になにも回答していないが，1.で受けた早くできることが期待されている印象が間違っていないことが読み取れる．実際，次回で動作検証というのは無理であったので，ベンダー側は定期的な進捗の間隔を少し密にしてみることを検討した．

6.は打ち合わせ後の落ち穂拾いである．

打ち合わせが終わって帰路，あるいは自分のデスクに戻ってから「あ，いま思いついたけど…」と言うのはよくあることなので必要であれば用紙の裏面を使って文章や絵で自由に書ける箇所とした．

このケースでは打ち合わせで出てこなかったクライアント側の要望をひとつ，拾い上げることができた．

4-2. 実施に際して

その後，複数回（この事例の他を含む）Minute Paperを配布，回収して気づいたことを述べる．

第一に回収されることが最重要の目標となる．回収されなかったということ自体もひとつの情報ではあるが（後日，その理由を確認する），忙しい，あるいは関心が薄く遂に回収がされないことが時々発生する．”特になければ「なし」で構わない”等と説明し，できるだけ回収に協力してもらえるように手を尽くす．解答方法は書面でもメールでも FAX でも…と柔軟に対処する．

非常に大事なこととして，当日の終業前か翌日中に記入してもらうように念を押しておくことである．2日より多くの日数が経過すると回収率ががくんと下がることが実施した事例から散見された．また，日数が経過すると打ち合わせ時に浮かんだ疑問や思いは簡単に霧消してしまうようで，その点からも早めの記入を求めるべきである．

また項目数は4- 6が適切と考えられる．A4用紙1枚で配布する形をとるが質問は表面にまとめた方がよい．実際，問6は文章や絵など自由に記入して欲しい問いであったので裏面にまわした事があったが，項目がより多く感じられ気後れしてしまうというクライアントの意見があった．解答欄がせますぎると感じる参加者は勝手に裏面なり別ルートで意見を伝えてくれることがある．

問題数を削るのは悩ましいことであるが，おそらくこの6項目は多いほうだという印象を現在の著者は持っている．

分析は一度の実施単位で回収したデータを表計算ソフト等を使ってベンダー側参加者，クライアント側参加者を共に横に並べる，複数回の実施データは同様に時系列で縦あるいは別シートに並べる．このようにすると活発な参加者・消極的な参加者がわかりやすい．消極的な参加者も開発の関係者でありその意見は「ない」のではなく共有されていない状態だということに留意する．

また横方向・縦方向に繰り返される意見の塊ができることがある．開発の進み具合でそれらは移り変わっていくが，ここに開発の進み具合以外の変数が隠れていることがあり，なにが発生しているか読み取る努力が求められる．

回答は上記のようにまとめられ，プロジェクトの「現在」として共有される．ただ，筆者は集まった分析は次の提示物やインタビューになんらかの形で必ず反映させることとして直接生データをクライアント側に現状公開してい

ない. 集まった回答の中にはプロジェクトの成否によって事業の継続が変わってくることや, 責任論などの問題や, 具体的な個人名を挙げた意見なども含まれるためである. いずれも大きな示唆を与えてくれる意見であるが公開には注意を要する. また, このようなアンケート形式の質問票の場合, 回答者は暗黙の了解としてそれが直接公開されることは想定していない. 公開する方針で実施する場合は Minute Paper 上にその旨, 明示しておく必要があるだろう.

回答には直接顔を合わせたインタビューでは言いにくかったこと, インタビュー後に考えを整理して初めて出てきた意見, 時に厳しい指摘などが この機会がなければずっと遅くなったであろう早いタイミングで出てくる. 適切にフォローしていくことが問題の早期発見や合意形成に役立つと考える.

4-3. 議事録との比較

開発現場ではさまざまな形で打ち合わせが繰り返されるが, その内容はメモとして保管されるか, もう少しまとまって議事録(Minute)として関係者に配られる.

議事録は多くの場面で作成されるので馴染みがあると思われる. ここで, 改めて議事録と Minute Paper の比較を行いたい.

議事録は既に行った打ち合わせで明らかになった内容をまとめ, その決定事項とそれに至る経緯を記して会議参加者および参加できなかった関係者に配布し保管されるもので, 良く書かれた議事録は第三者が見てもその流れが把握できるものになっている.

特に, 後から「言った・言わない」のトラブルが発生したときには非情に効果的に働くので, コミュニケーション不良を抱えているプロジェクトでは議事録の制作と配布で問題が改善される事柄も多い.

しかし, 会議を終えて参加者に回ってきた議事録はその後どう参加者に働きかけるだろうか. 参加者はデスクに回ってきた議事録を一瞥してファイルに綴じてしまわないだろうか. 保管された議事録が再び読まれるのは問題が発生したあとではないだろうか. ある人にとって打ち合わせは自分のデスクに戻ってきた時点で「終わった」ものになっていないだろうか.

つまり関心の薄いまたは議事録を常に確認する習慣のない参加者に対して, 議事録はなにも行わない. 参加者をコントロールするすべを持たない. 打ち合わせ中に覚えた違和感や疑問の行き先は確保されていない.

それに対して Minute Paper は打ち合わせで明らかにならなかったことを記録する文書である.

Minute Paper は打ち合わせを参加者がデスクに戻った時点で終了させない. これを使用することで参加者に「振り返って考える時間」を持つことを働きかける. 筆者の Minute Paper の場合, 参加者にとって「この打ち合わせで重要だった事」はなんだったのか. 言いそびれた事, 伝わったか不安なこと, 次回への期待などを意識してもらうようデザインした.

そして記入して提出するというアクションを求める. 自分が打ち合わせに参加しているという自意識を強化するとともに, プロジェクトは参加者を必要としているというシグナルを送る.

この点で Minute Paper は単なる議事録を超える「打ち合わせをデザインする」能動的な文書だと捉えられる.

5. 結論

本研究では, 要求工程の打ち合わせがベンダーとクライアントがその認識を確認する非常に重要な場であるということから, 打ち合わせをデザインしてその価値を高めるために能動的な文書としての Minute Paper を適用した.

4-1 の分析にみるように
・解釈の相違を電話で確認している
・定期的な進捗の間隔を少し密にするよう検討している(これは実際に実施された)
・クライアント側の要望をひとつ, 早期に拾い上げた
と, 打ち合わせ後の参加者の考察や試みが Minute Paper を通じて発言や行動として表現され, 参加者がどのように現状を認識しているかという事柄は次の打ち合わせの参考すべき情報として利用された.

Minute Paper は, その問いの空欄を埋めるために短くとも時間を取り, 振り返って考えることがデザインされている. また, 書くというアクションを定期的に求めることで関心の薄い参加者に参加を即し, また参加が求められているというシグナルを送る. これらは個人の内で考察や試みの時間を確保し発言として外部に表現することに他ならず参加意識の向上を果たしている.

ごく小規模な環境から適用可能であり, クライアント・ベンダーを問わず参加者全員の意見を横断的に見渡せる. その結果, 打ち合わせに潜んでいた個々人の関心や理解度を分析するソースとなる.

Minute Paper の分析結果は次回の提示物や打ち合わせの議題としてフィードバックされ, なにを共通認識として持っているのか, 合意を図るべき課題はなにかをというコミュニケーション不良に基づく問題を改善する.

現在は主に教育の分野で用いられる手法であるが, 以上のような有効な効果からソフトウェア開発にも必要な従来にない手法であると考える.

6. 今後の課題

事例を収集し，Minute Paper のどのような記述から現在何が発生していると読み取れたのか，またプロジェクト終了後に開発全体を振り返り，発生する問題が打ち合わせの際にどの時点でどのように兆候として表れていたかを Minute Paper から検出することが可能であるのかを集め，読み取る際のポイントとして分類・整理したい．

ただし，この「回答から読み取る」という分析の形は，読み手の着眼点や経験に左右されるのは否めない．多くの実施者による多様な分析は経験的なセオリーとして価値があるものと思われるが，昨今の質問紙調査の自由回答テキストからのデータ分析を参考により定量的な分析が行えないか試みることも考えたい．

それらの為には最初に述べたように事例の収集が必要なのだが，筆者個人が年間に携わるプロジェクトの数は限られており，どうしても多くの実施者による実例の報告が不可欠である．もちろんその結果は報告者にフィードバックして共に考えるものとしたい．
もし，本研究に興味を持っていただけたなら実施して筆者に連絡をいただけると幸いである．

また，その他アクティブ・ラーニング手法の中にも開発現場への応用の可能性が見いだせると考えており，その方面でも研究を継続したい．

参照文献

[1] 栗田香代子 他，“インタラクティブ・ティーチング，” 2015. [オンライン]. Available: https://lms.gacco.org/courses/course-v1:gacco+ga017+2016_04/about. [アクセス日: 15 05 2016].

付録　Minute Paper

場所	
日時	氏名

1. 本日の打ち合わせで最も重要だと感じたこと，印象に残ったことは何ですか？

2. 言いたいと思ったことはいえましたか？言いそびれたことがあればお書きください．

3. 相手の話は理解できましたか？　分からなかったことがあればお書きください．

4. 相手はあなたの話を十分理解してくれたと思いますか，まだ理解が不十分かもしれないと思われるところはありませんでしたか？

5. 次回の打ち合わせで改善して欲しい点，期待することはありますか？

6. 打ち合わせ後に思いついたアイデアなどがあればお書きください．(文または絵)

（必要であれば裏面もお使いください）

返送方法：手渡し,または Email:okamoto@nms.ne.jp , FAX:088-679-7099
この回答は今回のプロジェクトの改善および技法の研究に使用します。

「保守あるある診断ツール」による保守課題の可視化事例

室谷　隆
TIS株式会社
Muroya.takashi@tis.co.jp

要旨

近年のシステム開発は，既存のソフトウェアに追加，改修するといった保守開発が大半であり，生産性や品質の向上に向けたプロセス改善のニーズが高まっている．

プロジェクトの課題を可視化する方法としては，CMMIなどのアセスメントや，IPA/SECが提唱しているSPIN^{3}CHなどがあるが，課題を導き出すための専門家が必要であり，時間，コストも掛かるのが通常である．

この様な背景の中，私達は保守開発の課題を短時間で見付け出すためのツールを作成し，適用した．その結果，保守の課題を可視化するツールとして有効であると認識した．またその使い方に関しても興味深い知見が得られたのでここに事例を紹介する．

1. ツールの概要

今般作成し，社内で適用した「保守あるある診断ツール」はセルフアセスメントツールの一種であり，以下の特徴を持っている．

1. 短時間（約 30 分）で実施可能
2. 生産性や品質の低下を招いている問題事象をチェックするだけ
3. 保守開発を特徴付ける 8 つの視点を持つ

1.1. 設問の特徴

世の中に存在するセルフアセスメントツールは，アセスメントモデルのプラクティスが実施できているかを直接問うものが多く，プラクティスを理解していないと回答が難しく，時間がかかるものであった．

このため，設問の内容を身近で発生している問題事象とすることで回答し易くした．[1]

1.2. 保守開発を特徴付ける 8 つの視点

保守開発は新規開発とは違い，修正箇所は少ないが，その影響調査とテストに工数が掛かると言った特徴を持っている．この特徴を踏まえ，以下の 8 つの視点（プロセス）で保守を捉えた．

1. 依頼/受付
2. 要件定義
3. 影響調査
4. 設計
5. コーディング/単体テスト
6. 品質確認テスト
7. リリース
8. 運用

2. 適用結果

社内のいくつかの保守開発に適用した結果，以下の知見を得た．

- PJ の課題が明確になり，改善策が立案可能
- 改善の前後比較で改善効果を可視化
- PJ 内の立場が違う要員で比較し認識齟齬を可視化
- 同一部門内での PJ 比較により，改善策事例の取得
- 他部門と比較し弱い所，強い所を可視化
- お客様と比較し認識齟齬が判明．改善対応箇所の可視化と改善策への合意形成により，顧客満足度の向上効果も見込める

これらの知見は，本ツールが課題を可視化するだけでなく，お互いの認識齟齬を解決するコミュニケーションツール，改善効果を確認できる定点観測ツールとしても活用でき，保守のプロセス改善に有効なものであると認識できた．

本ツールはその仕組み上，主観に基づいた結果が可視化されるため，客観性に欠ける面が有り，他社や他部門等大きく異なる環境下で作業している人との比較には向いていない．しかし同じ部署や同じ環境下で働く人に対する定点観測ツールとして大きな効果が期待できる．

参考文献

[1] 独立行政法人　情報処理推進機構　技術本部　ソフトウェア・エンジニアリング・センター編，プロセス改善ナビゲーションガイド～自律改善編～，2013

記憶力に基づくプログラム理解容易性評価尺度の追加実験

村上 優佳紗
近畿大学大学院
m.yukasa@gmail.com

角田 雅照
近畿大学大学院
tsunoda@info.kindai.ac.jp

中村 匡秀
神戸大学大学院
masa-n@cs.kobe-u.ac.jp

要旨

ソフトウェア開発の生産性を高めるためには，理解しやすいソースコードを書くことは非常に重要である．そのため，ソースコードの複雑度を評価する指標が，これまで数多く提案されてきた．例えばサイクロマチック数やCKメトリクスはソースコードの複雑度を評価する代表的な指標である．ただし，複雑度が低い場合でも，必ずしもプログラムが理解しやすいとは限らない．プログラムの理解のしやすさは，理解のために必要とする記憶量の多寡に基づくと仮定し，プログラム理解容易性評価尺度が提案されている．この尺度では，人間の短期記憶は，大きさの限られたキューになっていると仮定し，プログラムを読む時に，覚えるべき変数の個数がキューの大きさの上限を超える場合，理解が難しくなるとしてプログラムの理解容易性を評価している．ただし，指標を評価する実験では，学生を被験者としている．学生は年齢やプログラム経験が比較的均質であり，学生以外の，より幅広い年齢の被験者に適用した場合，必ずしも評価尺度がプログラムの理解容易性を表しているとは限らない．そこで本研究では，大学の教員を被験者実験とし，この評価尺度がプログラムの理解容易性を適切に表すかを確かめた．学生 24 人と大学教員 8 人を被験者として実験を行った結果，評価尺度がプログラムの理解容易性を適切に表していることがわかった．

1. はじめに

ソフトウェア開発の生産性を高めるためには，理解しやすいソースコードを書くことは非常に重要である．ソースコードが理解しにくい場合，ソフトウェアの保守性が低下しやすく，その結果，ソフトウェア開発の生産性が低下したり，ソフトウェアの品質が低下したりする可能性がある．そのため，ソースコードの複雑度を評価する指標が，これまで数多く提案されてきた．例えばサイクロマチック数[4]やCK メトリクス[1]はソースコードの複雑度を評価する代表的な指標である．

ただし，複雑度が低い場合でも，必ずしもプログラムが理解しやすいとは限らない．文献[3][5]ではプログラムの理解のしやすさは，理解のために必要とする記憶量の多寡に基づくと仮定し，プログラム理解容易性評価尺度を提案している．この尺度では，人間の短期記憶は，大きさの限られたキューになっていると仮定し，プログラムを読む時に，覚えるべき変数の個数がキューの大きさの上限を超える場合，理解が難しくなるとしてプログラムの理解容易性を評価している．

文献[3][5]では，プログラム理解容易性評価尺度の妥当性を確かめるために，学生を被験者として実験を行っている．ただし，学生は年齢やプログラム経験が比較的均質であり，学生以外の，より幅広い年齢の被験者に適用した場合，必ずしもこの評価尺度がプログラムの理解容易性を表しているとは限らない．そこで本研究では，大学の教員を被験者実験とし，この評価尺度がプログラムの理解容易性を適切に表すかを確かめる．

2. プログラム理解容易性評価尺度

プログラムの理解容易性を評価するために，文献[3][5]では，人間の記憶力に基づく尺度を提案している．これらの尺度では，プログラムを理解するためにメンタルシミュレーション[2]が行われることを前提としている．メンタルシミュレーションとは，プログラムの動作をコンピュータや筆記具を用いずに，思考しながら理解することであり，比較的規模の小さなコード片に対して用いられる．メンタルシミュレーションを行う場合，変数の値を記憶しておく必要があるが，多数の変数の値を同時に記憶しておくことは容易ではない．このため文献[3][5]では，多数の変数の値を記憶しておく必要のあるプログラムの場合，メンタルシミュレーションのコストが高くなり，プログラムの理解容易性が低下すると仮定している．

文献[5]では，人間の短期記憶を FIFO キューとして，メンタルシミュレーションの仮想モデル（Virtual Mental Simulation Model; VMSM）を作成し，そのモデルに基づいて理解容易性評価尺度を定義している．基本的なアイディアは，ある変数の値を参照する際，（大きさに制限のある）キューに変数の値が記憶されている場合はコストが小さくなるとしている．逆にキューに値が記憶されていない場合，その値の変更箇所にまでバックトラックする必要が生じるため，コストが大きくなるとしている．

文献[5]では，VMSM に基づき，以下の 4 つの評価尺度を定義している．

- ASSIGN: 変数代入に関するコスト．
- RCL: 短期記憶内の変数を思い出すコスト．
- BT_CONST: 定数をバックトラックする回数．短期記憶にない定数を得るコスト．
- BT_VAR: 変数のバックトラックの距離．短期記憶にない変数を得るコスト．

文献[3]では，上記のメトリクスが変数の更新回数に基づく再計算コストを考慮していないことと，バックトラックに関するメトリクスがプログラムの行の入れ替えに敏感すぎることが問題点であるとし，新たな評価尺度を 2 つ提案している．具体的には，各変数の値の更新回数をベクトルの要素とし，ベクトルの要素の和に基づくメトリクス SUM_UPD とベクトルの要素の分散に基づくメトリクス SUM_VAR を定義している．

3. 実験

3.1. 概要

実験の目的は，学生以外（今回の実験では大学教員）の場合でも，プログラム理解容易性評価尺度が妥当に機能するかどうかを確かめることである．そのために，必要とする記憶力が異なる，複数のプログラムを用意し，被験者がコードを理解するために掛けた時間を計測した．

必要とする記憶量の異なるプログラムは，石黒らの研究[3]で示されているもの 4 つ（a0, a1, b0, b1．図 1，図 2 参照）を利用した．各プログラムは 20 行から 30 行の規模である．あらかじめ指定された変数の値が，プログラム実行後にどうなるかプログラムを読んで答え，それが正しかった場合，プログラムを理解できたとした．例えばプログラム a0 の場合，プログラム実行後の変数 i の値を答えさせた．プログラムの理解はメンタルシミュレーションにより行うこととし，メモなどは利用させないようにした．

各プログラムで理解のために必要とする記憶力の多寡

```
int i , t;

t = 11;
t = t - 1;
i = 2;
if(i < t){
        i = i + 2;
        if(i < t){
                i = i + 2;
        }
        if(i < t){
                i = i + 2;
                if(i < t){
                        i = i + 2;
                }
        }
        if(i < t){
                i = i + 2;
        }
}
System.out.println("i = " +i);
```

(a0)

```
int i , t;

t = 11;
t = t - 1;
i = 2;
if(i < t){
        t = t - 2;
        if(i < t){
                i = i + 2;
        }
        if(i < t){
                t = t - 2;
                if(i < t){
                        i = i + 2;
                }
        }
        if(i < t){
                i = i + 2;
        }
}
System.out.println("i = " +i);
```

(a1)

図 1 プログラム a0, a1[3]

```
int a, b, c, d, e, f, g;

a = 2;
b = 4;
c = 3;
d = 6;

c = c + 4;
d = d - 2;
if(c < 5)
        e = d + 5;
else
        e = d + 3;
a = a * 2;
b = b + 6;
if(a > 7)
        f = b - 3;
else
        f = b - 5;
g = e + f;

System.out.println("g = "+g);
```

(b0)

```
int a, b, c, d, e, f, g;

a = 2;
b = 4;
c = 3;
d = 6;

c = c + 4;
d = d - 2;
a = a * 2;
b = b + 6;
if(a > 7)
        f = b - 3;
else
        f = b - 5;
if(c < 5)
        e = d + 5;
else
        e = d + 3;
g = e + f;

System.out.println("g = "+g);
```

(b1)

図 2 プログラム b0, b1[3]

については，2 章で説明した 6 つのメトリクス（ASSIGN, RCL, BT_CONST, BT_VAR, SUM_UPD, VAR_UPD）に基づき評価した．これらのメトリクスに基づく各プログラムの理解容易性を表 1に示す（石黒らの研究[3]からの引用）．ASSIGN, BT_CONST, SUM_UPD より，プログラム b0, b1 を理解するためには，比較的記憶力が必要とされることがわかる．

被験者を学生グループと教員グループに分け，それぞれのグループの解答時間の平均値や中央値などを算出

表 1　各プログラムの理解容易性[3]

プログラム	ASSIGN	RCL	BT_CONST	BT_VAR	SUM_UPD	VAR_UPD
a0	12	6	0	80	7	1.25
a1	12	6	0	48	7	0.25
b0	18	8	1	30	11	0.24
b1	18	6	1	54	11	0.24

し比較した. 学生グループは, 近畿大学理工学部情報学科に所属する学部 4 年生から修士 2 年生の 24 名であり, 教員グループは同学科の教員 8 名である. 教員グループの平均年齢は 45 歳（最小 33 歳から最大 64 歳）である.

プログラムを読む順番が実験結果に影響することを避けるために, 4 つのプログラムを読む順番を被験者ごとに変更した. 例えばある被験者ではプログラムを a0, b1, a1, b0 の順に読むとし, 別の被験者では b1, a0, b0, a1 の順に読むなどとした.

3.2. 実験ツール

実験のために, 問題を出題し, 回答時間や誤回答の回数を計測するためのツールを作成した. 図 3 にツールのスクリーンショットを示す. 本ツールは表計算ソフトのマクロを用いて開発した. ツールの動作を以下に示す.

1. 「解答する」をクリックすると問題とテキストボックスを表示する.
2. 問題に正解するまでテキストボックスを表示し続ける.
3. テキストボックス（問題）が表示されてから正答するまでの回答時間と誤回答の回数を記録し, 被験者にも表示する.
4. 正解すると次の問題に自動的に遷移する.

3.3. 結果

表 2 に各グループの回答時間の平均値と中央値を示す. 教員グループ, 学生グループとも, 記憶力を比較的必要としないプログラム a0, a1 と比べ, プログラム b0,b1 のほうが, 解答時間が長くなっていた.

比較的記憶力を必要としないプログラム a0 と比べ, 何倍の回答時間が掛かっているかを調べた. 具体的には, プログラム a0 以外の回答時間÷プログラム a0 の回答時間を求めた. この値が大きいほど, a0 と比較して回答時間が多く掛かっていることを示している. 結果を表 3 に示す. 学生グループでは, b0 / a0, b1 / a0 の平均値と中央値のうち, b0 / a0 の平均値以外は 2 を下回っていたが, 教員グループでは, それらの値全てが 2 を上回ってい

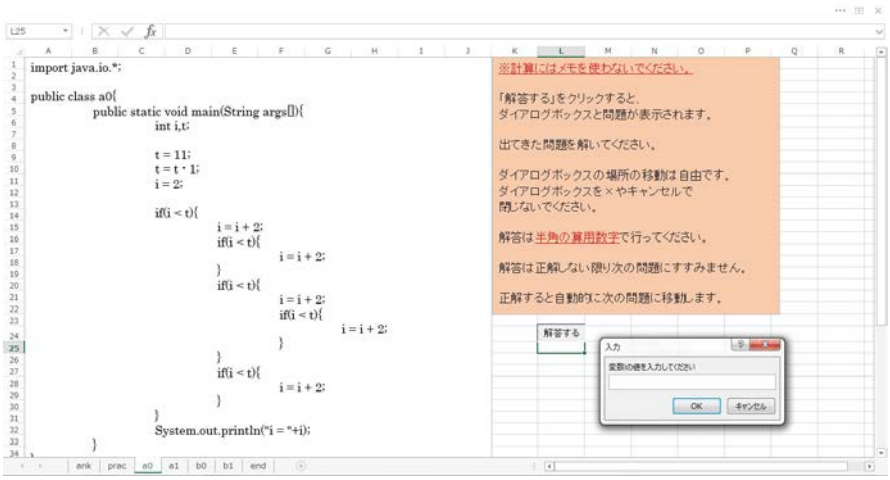


図 3　実験用ツールのスクリーンショット

表 2　各グループの回答時間（秒）

		a0	a1	b0	b1
学生	平均値	65	82	105	89
	中央値	58	75	100	70
教員	平均値	59	91	141	115
	中央値	51	51	144	109

表 3　プログラム a0 との回答時間の比

		a1 / a0	b0 / a0	b1 / a0
学生	平均値	1.6	2.0	1.4
	中央値	1.2	1.5	1.4
教員	平均値	1.5	2.4	2.2
	中央値	1.0	2.4	2.4

表 4　誤回答数の基本統計量

		a0	a1	b0	b1
学生	平均値	1.0	0.5	1.3	0.8
	中央値	0.0	0.0	0.0	0.0
	標準偏差	2.5	0.8	2.2	2.3
教員	平均値	0.1	0.4	1.1	0.5
	中央値	0.0	0.0	1.0	0.0
	標準偏差	0.4	0.7	1.0	0.8

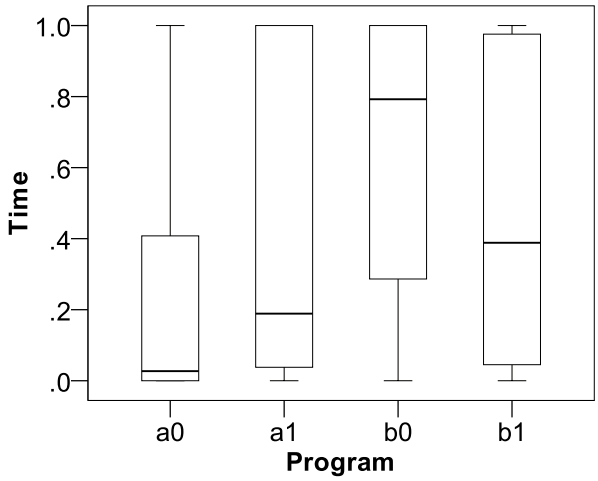


図 4　学生グループの回答時間正規化

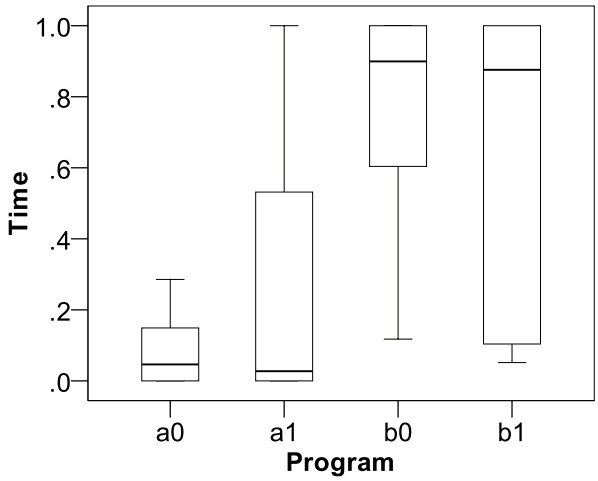


図 5　教員グループの回答時間正規化

た.

被験者ごとに解答時間を正規化して比較した. 正規化は(回答時間 – 最小解答時間)÷(最大回答時間 – 最小回答時間)により行った. 正規化を行うことで被験者個人の問題の得手不得手を判別することができるためである. 箱ひげ図を図 4, 図 5 に示す. 学生グループの場合, プログラム a0 以外はばらつきが大きい, すなわち記憶力を比較的多く必要とするプログラム b0, b1 の場合でも, 相対的に回答時間が長くなるとは限らないといえる. 教員グループの場合, 特にプログラム b0 において相対的に回答時間が長い傾向があった. これらの結果より, 教員グループのほうが, プログラム理解容易性評価尺度がより明確に機能しているといえる.

誤回答数の基本統計量を表 4 に示す. プログラム b0 を除き中央値は 0 であり, どちらのグループも誤回答を多く行った被験者は少ないことがわかる. プログラム b0 については, 教員グループの中央値が 1 となったことから, このプログラムは教員グループにとって, 特に理解が難しかったと考えられる. なお, 学生グループでは, 一部の被験者の誤回答数が多く, プログラム a1 を除き, 標準偏差が 2 を超えていた. ただし, 教員グループと比較して, 平均値と中央値に大きな差はないため, 全体としては正答率に教員グループと差はないといえる.

表 5　アンケート結果

	問 I			問 II		
	1	2	3	1	2	3
学生	43%	43%	14%	33%	50%	17%
教員	57%	29%	14%	46%	46%	8%

3.4. 考察

教員グループの被験者が，実験に集中して取り組んでいなかった可能性は低いと考えられる．これは，プログラム a0 の出題順序が被験者によって異なっていたにも関わらず，a0 の回答時間が学生グループよりも低かったためである．

実験後のアンケートで出題したプログラムについてアンケートを行った．その内容は以下の 2 つである．

I. 1 問目と 3 問目について，具体的にどの部分がどのように変更されたかについて気づいたか．
II. 2 問目と 4 問目について，具体的にどの部分がどのように変更されたかについて気づいたか．

回答は以下の 3 つとした．

1. 気づいたうえで早く読めたと思う
2. 気づいたが早く読むのに役立たなかった
3. 気づかなかった

学生グループの有効解答数は 24，教員グループの有効解答数は 7 であった．アンケート結果を表 5 に示す．学生グループと比較して，教員グループは問 I，問 II とは「1:気づいたうえで早く読めたと思う」と解答した被験者が多かった．図 5 において，教員グループのプログラム a1 の回答時間のばらつきが比較的小さかったのは，このことが影響している可能性がある．記憶力を必要とするプログラム b0，b1 について，もし回答 2，3 が多かった場合，回答時間がより長くなった可能性がある．

4. おわりに

本研究では，記憶力に基づくプログラム理解容易性評価尺度[3][5]に着目し，指標の妥当性を評価するために，被験者の対象を変えて実験を行った．この尺度では，人間の短期記憶は，大きさの限られたキューになっていると仮定し，プログラムを読む時に，覚えるべき変数の個数がキューの大きさの上限を超える場合，理解が難しくなるとしてプログラムの理解容易性を評価している．文献[3][5]の実験では，評価尺度の妥当性を確かめるために，学生を被験者として実験を行っている．

ただし，学生は年齢やプログラム経験が比較的均質であり，学生以外の，より幅広い年齢の被験者に適用した場合，必ずしもこの評価尺度がプログラムの理解容易性を表しているとは限らない．そこで本研究では，大学の教員を被験者実験とし，この評価尺度がプログラムの理解容易性を適切に表すかを確かめた．その結果，学生の被験者よりもより明確に，プログラム理解容易性評価尺度が適切に機能した．

謝辞　本研究の一部は，文部科学省科学研究補助費（挑戦的萌芽：課題番号 26540029，基盤 C：課題番号 25330090）による助成を受けた．

参考文献

[1] Chidamber, S. and Kemerer, C: A metrics suite for object oriented design, IEEE Transactions on Software Engineering, vol.20, no.6, pp.476-493, 1994.
[2] Dunsmore, A., Roper, M. and Wood, M.: The role of comprehension in software inspection, Journal of Systems and Software, vol.52, no.2–3, pp.121-129, 2000.
[3] 石黒誉久，井垣宏，中村匡秀，門田暁人，松本健一：変数更新の回数と分散に基づくプログラムのメンタルシミュレーションコスト評価，電子情報通信学会技術報告，SS2004-32，pp.37-42, 2004.
[4] McCabe, T: A Complexity Measure, IEEE Transactions on Software Engineering, vol.SE-2, no.4, pp.308-320, 1976.
[5] Nakamura, M., Monden, A., Satoh, H., Itoh, T., Matsumoto, K., and Kanzaki, Y.: Queue-based Cost

Evaluation of Mental Simulation Process in Program Comprehension, Proc. of International Software Metrics Symposium, pp.351-360 (2003).

チームの協働状態を測る：Team Contribution Ratio
～手間のかかる質問紙からの脱却～

増田 礼子
フェリカネットワークス
Ayako.Masuda@FeliCaNetworks.co.jp

森本 千佳子
東京工業大学
morimoto@cs.titech.ac.jp

松尾谷 徹
デバッグ工学研究所
matsuodani@debugeng.com

津田 和彦
筑波大学大学院
tsuda@gssm.tsuka.tsukuba.ac.jp

要旨

本論文では，ソフトウェア開発チームの特性を，容易に計量するための指標を構築することを目的とする．ソフトウェア開発に影響を与えるチーム特性には，リーダシップやコミュニケーションなど，さまざまな要因が考えられる．チーム特性の計量には，心理尺度を応用した質問紙を用いた方法が使われている．質問紙による計量は，大量の調査データを統計処理する必要がある．そのため，質問紙を用いた計量は，実務現場においては敷居が高く，個々のチーム特性の把握を容易に行うことができないという課題がある．

本稿では，チーム特性の要素のひとつである協働を対象とし，構築した指標であるチーム貢献係数 *(Team Contribution Ratio)* を用いた計量を実施し，その内容を報告する．チーム貢献係数は，経済学において所得の均等度合いを表すローレンツ曲線と *Gini* 係数を応用した指標である．協働の分析対象は，チームメンバそれぞれが持つ技術，知識，経験を開示し，共有するための知識ベース構築活動とした．具体的には，協働の道具として用いたグループウェアの投稿数と投稿者数を基にした計量である．

1. はじめに

ソフトウェア工学において，ソフトウェア生産性に開発チームの状態が影響を与えていることを示したのは，Bohem のコストモデル (COCOMO) に関する一連の研究である．コストモデルの研究は，ソフトウェア規模とコストの関係を生産性と定義し，その変動要因を生産性因子として計量化を行い，大規模な調査を行った．「Software Engineering Economics」にまとめられたこの研究の結果は，多くの生産性因子の中で最大の影響因子が「チームの能力」であることを示した [1, 2]．この研究では「チームの能力」を計量するため，複数の管理者がスコアリングを行い，決定している．

この研究を追証し，実用化を試みた国内通信メーカにおける大規模な調査と分析では,「チームの能力」を 3 つの因子に分解し，計量にバイアスがかからない工夫を行った．その結果，同様に「チームの能力」が影響していることが確認されている [3, 4].

ソフトウェア分野において，生産性が「技法やツール」,「プロセス」に依存することはよく知られており，さまざまな研究と実用化が行なわれている．しかし,「チームの能力」に関する研究や実用化は，他の分野と比較すると遅れている．他の分野，たとえば製造業では「チームの能力」を「現場力」と呼び，国際競争力の根源と位置付けられている．ソフトウェア分野におけるチームの計量化については，従業員満足の考え方を日本のソフトウェアチーム特性に合わせて拡張し，大規模な計量を行ったパートナー満足調査がある [5, 6, 7]．また，この研究結果を応用し，プロジェクトの成功失敗を判別する研究 [8] やチーム・パフォーマンスの比較研究 [9] が行われている．

これまでの研究における計量手段は，心理尺度における構成概念に基づいて設計された質問紙を用いる方法が一般的である [10, 11]．しかしながら，質問紙の設計には，心理学における高度な技術が必要となり，現場での実用は困難であるという課題が指摘されている．

本論文の研究目的は，質問紙調査より容易，かつ客観的に計量するための指標を構築することである．ソフトウェア開発における協働は，単純な作業の分担ではなく，知的な活動である．定められた期間の中で，設計，実装，テストへと開発を進めていく過程では，メンバそれぞれが持つ技術，知識，経験の伝達や共有が重要となる局面は多い．そこで，チーム特性のひとつである協働に注目した．計量を容易にすることが目的であるため，観測しやすく，定量的な成果物を分析対象とする必要がある．協働状態が促進すると，チームでの情報共有が活発になり，知識の相互提供が増加すると考えた．このことから，チームによる知識ベース構築活動の成果物を基に，事例分析を行った．具体的には，知識共有を目的としたグループウェアの投稿数の経年変化の比較分析である．

比較研究を容易にする方法として，経済学において，国や地域における所得格差を分析するローレンツ曲線と Gini 係数が良く知られている [12, 13]．ローレンツ曲線と Gini 係数は応用範囲が広く，さまざまな格差の研究に応用されている．本稿では，チームの協働状態を計量する指標として，ローレンツ曲線と Gini 係数に着目し，値が大きいと良い指標値となるよう調整を行い，Gini 係数と区別するためチーム貢献係数と名付けた．

チーム特性については，仲間意識や規範意識など，いくつかの主成分が知られており [8, 9]，これらの要素を比較するために，どのような成果物を用いて，どのように比較するのかが課題となっている．ここでは，チーム特性のひとつの要素であるチームの協働状態について，実際に計量を行い貢献係数を示し，その値の変化や差について考察を行う．また，今後の応用方法や効果について検討する．

2 章では，貢献係数とその計算方法，対象と母集団について定義する．3 章では，事例に対し，提案する方法で行ったチームの協働状態の計量結果を示す．4 章では，貢献係数について検討し，今後の計量の提案を行う．

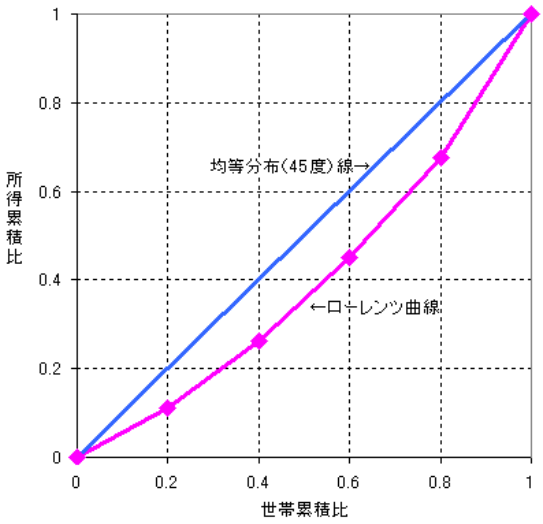


図 1. 世帯所得のローレンツ曲線の例

表 1. 世帯数累積比と所得累積比

世帯構成比	0	0.2	0.4	0.6	0.8	1
所得構成比	0	0.111	0.264	0.45	0.677	1

2. チーム貢献係数の対象と計量方法

チーム特性の計量における課題は,「何を対象に」,「どのような計量を行えば」,「何がわかるのか」に分解できる．この章では，チーム貢献係数についてこの視点からまとめる．

2.1. 所得の均等度合いを表すローレンツ曲線

先に示したように，経済学では，国家や地域における所得の均等度合いを表す手法としてローレンツ曲線 (Lorenz curve) と Gini 係数 (Gini inequality index, Gini ratio) がよく知られている [14]．ローレンツ曲線の例を図 1. に示す．

図 1. の縦軸は所得の累積比率で，横軸は世帯累計比率である．用いたデータは，表 1. 世帯数累積比と所得累積比に示した値である．均等分布 (45 度) 線は，所得が均等に分布した場合を表している．

Gini 係数は，ローレンツ曲線と均等分布線によって囲まれる領域の面積を，均等分布線より下の領域の面積で除算した値である．均等分布線より下の領域は，三角形となるため，その面積は 1/2 となる．よって，Gini 係数はローレンツ曲線と均等分布線によって囲まれる領域の

面積を 2 倍したものになる．Gini 係数は，完全に均等分布の場合の値は 0 となり，不均等になるにつれて値が大きくなる指標である．この値を用いて，所得格差を量ることができる．

2.2. コミュニティの計量

北山は，組織内コミュニティの計量分析に，メーリングリストの発信数をコミュニケーション量として，ローレンツ曲線と Gini 係数を用いて分析を行った [15]．また，コミュニティの人数と Gini 係数の相関を調べている．

ネットワーク上のデータの分析に Gini 係数 (Gini index と呼んでいる）を用いた研究もある [16]．この研究では，Web の閲覧行動における特定のサイトに集中する程度を Gini 係数で表現し，比較している．

このように，ローレンツ曲線と Gini 係数は，本来の目的である所得の均等度合いを表す以外にも活用されている．この研究の事例の通り，Gini 係数は所得均等性の評価指標として提案された係数であるが，均等性を評価するさまざまな事項へ応用が可能である．

2.3. チーム貢献係数 (Team Contribution Ratio)

本論文の課題は，チーム特性を計量し，比較研究を行うことである．この手段として Gini 係数を用いるが，本来の使い方とは異なるので再定義を行う．

「何を対象にするのか」については，チームの活動を対象とする．チームとは，組織行動論の定義によれば，同じ目的を共有する集団である [17]．ソフトウェア開発におけるプロジェクトも，規模に係わらずチームと見なすことができる．

「どのような計量を行うのか」については，本来の Gini 係数が世帯における収入を対象としたのに対して，本研究での提案は，チームの協働活動の成果物を対象とする．どのような成果物を対象にすれば，グループの状態の何を表すと言えるのかについては，さらなる研究が必要である．計量化としては，対象を構成要素に分け，累積比を求めることができれば良い．成果物の構成要素は，必ずしもチームのメンバとは限らない．たとえば，開発チームの成果物として文書類を対象とし，その種類として仕様，テスト関連，インタフェースなど種類に分けて計量することもある．この時の貢献係数も，チームのある種の特徴を表すと考えられる．

Gini 係数との違いとして，貢献係数は値が大きい方が，広く貢献していることを表す．そのため，図 1 のローレンツ曲線と均等分布線との間の面積ではなく，ローレンツ曲線下の領域の面積を用いる．具体的には「貢献係数 =1-Gini 係数」と定義できる．

2.4. 個人のロールとチーム貢献

チームの何を対象に計量するのかについては，注意が必要である．たとえば，営業を行うチームにおいて，営業担当者の個々の売上を対象にローレンツ曲線と貢献係数を求めたとする．研究者は，営業チームとして協働状態を計量するのが目的であったとしても，営業チームのメンバである各営業担当者は，売上競争をあおるノルマ管理と受け取る可能性がある．

人を対象とする計量は，思わぬ副作用を伴うので注意が必要である．容易に計量できるようになることで，本来の目的とは異なる目的や解釈で利用されてしまう危険性が伴う．この点については，計量方法の提案と共に，利用用途や活用方法についても慎重に議論をし，注意を促していかなければならない．

3. 貢献係数の計量事例

ここでは，ある事例チーム (チーム X) に対し，チーム貢献係数を用いて計量し，特徴を抽出できるかについて検討する．具体的には，グループウェアへの投稿数からチームの協働状態の計量を試みた．

ソフトウェア開発を行っているチーム X は，プロジェクトマネジャーを中心に，いくつかのサブチームに分かれて開発を進めていた．開発フェーズに合わせてサブチームの構成は臨機応変に組み替えが行われていたこと，開発を進めていくにあたってサブチーム間でのさまざまな情報共有が必要であったことから，チーム全体として，技術，知識，経験の蓄積と共有を求める声が高まった．そして，知識ベース構築に向け，グループウェアの運用が開始された．分析対象のグループウェアは，メンバによる自主的な運用が行われており，かつ，技術的な内容を取り扱っているものである．グループウェアにおける計量は，投稿者数と投稿数を対象とした．

投稿の形態が異なる 2 つのグループウェアを計量対象とし，Wiki [18] ベースのものを A，Moodle [19] ベースのものを B と呼称する．A の特徴は，投稿 1 件がひと

つの情報であり，Wiki のコメント機能を用いて運用されていた．B の特徴は，Moodle のフォーラム機能を用いており，基本的には質問が投稿され，それに対する回答が投稿される形で運用されていた．どちらも分量に関わらず 1 投稿を 1 件として計量した．ただし，B においては，必ずしも質問・回答形式となっているわけではなく，情報提供としての投稿のみのものもあれば，自分で質問して自分で回答するという自己レス形式での投稿もあった．しかし，いずれも 1 項目に複数の情報は混ざっていない．また，質問と回答の種別による重み付けは実施していない．グループウェアの投稿時には，投稿者の名前が表示されるため，導入当初は特に初歩的な質問の投稿を躊躇するメンバが多かった．B の活用状況の調査を目的とした無記名のアンケート調査では，「こんなことを質問して良いのか」，「自分は知らなかったが，みんなは既に知っている情報ではないか」，「そんなことも知らなかったのかと思われるのではないか」等の声が寄せられた．しかし，投稿を躊躇するメンバであっても，情報の共有と蓄積の重要性は認識しており，「今後のために情報を蓄積していこう」，「誰かの役に立つからどのような情報でも書き込もう」という働きかけと共に，投稿者が増えてきた．メンバが感じる投稿への心理的ハードルは，質問・回答のいずれであっても同じであった．これが，重み付けを行わない理由である．

3.1 節と 3.2 節において，チーム X における 2012 年と 2014 年のグループウェアへの投稿数を基にした計量結果を示す．3.3 節では，計量結果を基に，2012 年から 2014 年への時系列変化について考察する．

3.1. グループウェア投稿を基にしたチーム貢献係数 (GW2012)

本節で対象としたグループウェア (GW2012) は，主に仕様変更による影響分析に関する情報を共有するために 2012 年に運用していたものである．運用には，Wiki を用い，半年間に渡り，チームメンバへ情報提供を呼びかけていた．

計量対象期間は 6ヶ月，対象期間のチームメンバは，42 名であった．この期間の投稿者数は 11 名であり，その投稿数と投稿者数を表 2. に示す．

実際には，表 2. の数値に，未投稿者の 31 人分を投稿数 0 として追加し，母数をチームメンバ数と同数になるようにしている．このデータを用い，ローレンツ曲線と Gini 係数を，統計パッケージ R を用いて求めた．GW2012 における投稿状況のローレンツ曲線を図 2. に示す．

表 2. GW2012 の投稿数

投稿者	A	B	C	D	E	F	G	H	I	J	K
投稿件数	33	37	26	7	4	3	3	2	1	1	1

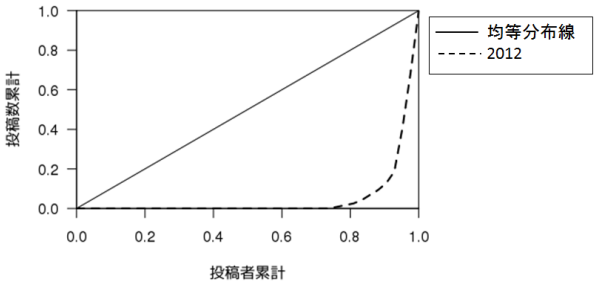


図 2. GW2012 のローレンツ曲線

この時期の投稿者は，全体の約 25%であり，チーム貢献係数は，0.10 であった．

3.2. グループウェア投稿を基にしたチーム貢献係数 (GW2014)

本節で対象としたグループウェア (GW2014) は，仕様変更の影響に限らず，テストや保守なども含めた業務全般の双方向の情報共有へと拡張するため，2014 年から運用を開始した．運用には，利用者インタフェースに優れた Moodle を用いている．

計量対象期間は，16ヶ月，対象期間のチームメンバは，24 名であった．この期間の投稿数と投稿者数を表 3. に示す．

表 3. GW2014 の投稿数

投稿者	A	B	C	D	E	F	G	H	I	J
投稿件数	33	26	21	16	15	9	7	6	5	4
投稿者続き	K	L	M	N	O	P	Q	R	S	T
投稿件数	4	4	3	3	3	2	2	2	1	1

3.1 節の GW2012 の場合と同様に，表 3. の数値に，未投稿者の 4 人分を投稿数 0 として追加し，母数をチームメンバ数と同数になるようにしている．表 3. のデータを用い，GW2012 と同様にローレンツ曲線と Gini 係数を求めた．GW2014 における投稿状況のローレンツ曲線を図 3. に示す．

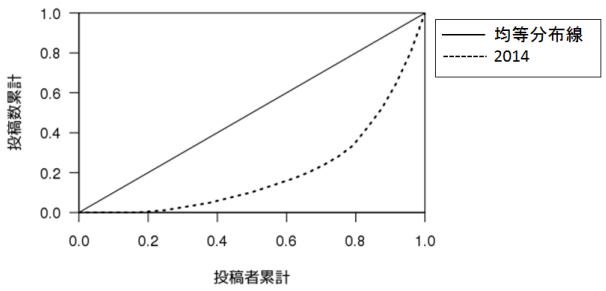


図 3. GW2014 のローレンツ曲線

この時期の投稿者は，全体の約 80%を占めるに至っており，チーム貢献係数は，0.40 であった．

3.3. 時系列変化の考察

GW2012 と GW2014 のローレンツ曲線とチーム貢献係数の比較を図 4. と表 4. に示す．

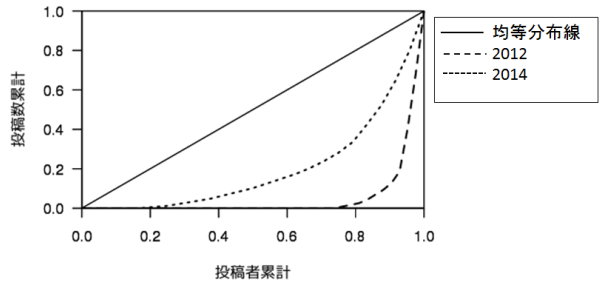


図 4. GW2012 と GW2014 のローレンツ曲線の比較

表 4. GW2012 と GW2014 の貢献係数比較

GW2012 の貢献係数	0.1
GW2014 の貢献係数	0.4

図 4. と表 4. は，GW2012 と GW2014 の対象期間におけるチーム X のグループウェアへの投稿数という観点から，チーム貢献係数を用いてチームの協働状態を表したものであり，両者の差は明らかである．チーム貢献係数が GW2012 から GW2014 での増加が何に起因しているのか，その原因の分析が，現在の課題となっている．しかし，今回調査した 2012 年と 2014 年のチーム X のグループウェアへの投稿数から見ると，チームの協働状態に何らかの変化があり，その変化を捉えることができていると考える．

4. 計量方法の比較

ここでは，質問紙によるチームの調査と，チーム貢献係数による計量について比較検討する．

4.1. 質問紙のコストと信頼性

一般的な質問紙による調査とは，社会調査の手法であり，同じ質問を複数の人に対して行うことにより意見を明確化する．チームに対する質問紙調査は，心理学における質問紙法と呼ばれるもので，社会調査のアンケートとは異なる．質問紙法は，人の心理的要素を「構成概念」と呼び，直接観測が困難な概念として取り扱う [10, 11]．たとえば，チーム研究で使われている「仲間意識」や「規範意識」と呼ぶものは，構成概念の一種である．

計量は，構成概念の要素を考え，間接的に観察できる質問項目に展開して行う．「仲間意識」であれば，「同じ仲間ともう一度仕事をしたいと思いますか」などの複数の質問項目から構成される．質問項目は，構成概念の下位尺度と呼ばれている．

質問紙法の品質は，2 つの特性で評価されている．一つは，「安定して測ることができるのか」，つまり，再現性があるか否かである．質問紙法では，この再現性のことを信頼性と呼んでいる．

もう一つは，「測りたいものを測ることができているのか」であり，質問紙法では妥当性と呼んでいる．測りたいものは，直接見ることができない構成概念であり，妥当性の検証は，統計的な手法を用いることになる．

質問紙法を使った計量手順を図 5. に示す．

質問紙法は，簡単に作成することはできない．チーム特性の計測のための構成概念の開発は，2000 年から始め，今日に至っている．現在までに 5,000 件を超える調査と分析，改善によってメンテナンスされている．

4.2. チーム貢献係数のコストと信頼性

チーム貢献係数の品質について検討する．質問紙法と同様に考えるなら，「信頼性」と「妥当性」である．「信頼性」，すなわち再現性については，質問紙法とは比較にならないほど確実である．これは，物理的に計量できる成果物を対象としているからである．

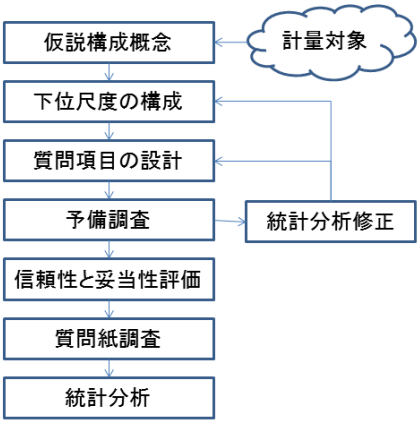


図 5. 質問紙の設計手順の例

「妥当性」，すなわち測りたいものを測ることができているのかについては，検討が必要である．3 章の事例で用いた，協働活動の結果であるグループウェアへの貢献度は，投稿数そのものを測りたいのではなく，チームにおける知的な援助行動を表すと考え，その代用特性として計量している．「妥当性」は，項目を詳細化して分解すれば達成できるが，問題は統合的な構成概念として計量できているのかについて評価する必要があるということである．チーム貢献係数による計量は，始まったばかりであり，まだ体系化が不足している．質問紙の質問項目と対応させてチーム貢献係数を論じれば，コスト的にも信頼性としても優れていることは確かである．しかし，構成概念レベルでの計量となると，まだ未完成であり，今後の研究と発展を待つ必要がある．

5. おわりに

本論文では，チーム特性の計量に関し，従来の質問紙法に代わる容易な計量方法について示した．その第一段階として，チーム特性のひとつの要素である協働について，経済学で用いられているローレンツ曲線と Gini 係数を応用したチーム貢献係数による計量方法を提案した．チーム貢献係数を用いて実際に計量を行い，その信頼性とコストについては良い結果を得ることができた．

調査の妥当性については，まだ検証過程であり，本稿の事例における計量結果の検証が次の課題である．現状では，まだ，調査項目単独のレベルであり，今後，構成概念の構造に当たる検討が必要である．

引き続き,「何を対象に」,「どのような計量を行えば」,「何がわかるのか」について，さまざまな成果物や事例に対して試行していきながら，検証を進めていく．そして，構成概念としての計量ができているのかを評価していく必要がある．今後解決しなければならない課題はあるが，チームの協働状態を容易に計量する方法として有用であると考える．

謝辞

本研究の実施にあたり，野村マネジメント・スクールの助成金のサポートを受けました．ここに記して感謝いたします．

参考文献

[1] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[2] B.W. Boehm, “Software engineering economics”, Software Engineering, IEEE Transactions on, vol.SE-10, no.1, pp.4–21, Jan. 1984.

[3] 寺本雅則，松尾谷徹，上村松男，“ソフトウェア開発過程を定量的に解析して生産性と品質を向上させるソフトウェアメトリクス”, 日経エレクトロニクス 6 月 4 日号，日経エレクトロニクス，1984.

[4] 松尾谷徹，“開発コストのメトリクスの現状と課題”, 昭和 63 年電気・情報関連学会連合大会予稿集，sep 1988.

[5] 松尾谷徹，“パートナ満足によるソフトウェア生産性の向上”, 第 20 回ソフトウェア生産における品質管理シンポジウム，日本科学技術連盟，Nov. 2001.

[6] 松尾谷徹，“パートナー満足と人的リソースのパフォーマンス”，プロジェクトマネジメント学会誌，vol.4，no.1，pp.3–8，2002.

[7] 榎田由紀子，松尾谷徹，“Happiness & active チームを構築する実践的アプローチ：チームビルディングスキルの開発”, プロジェクトマネジメント学会誌，vol.7，no.1，pp.15–20，Feb. 2005.

[8] 松尾谷徹，“It に現場力は存在するのか:その計測と評価の試み”, ソフトウェア・シンポジウム 2014in 秋田 SEA: ソフトウェア技術者協会，2014.

[9] 増田礼子，森本千佳子，松尾谷徹，津田和彦，“チームビルディングのメトリクス: ～ 10 年間の活動成果を測る～”, ソフトウェア・シンポジウム 2015in 和歌山 SEA:ソフトウェア技術者協会，2015.

[10] 小塩真司，西口利文，“質問紙調査の手順, 47-53”, 2009.

[11] 山本眞理子，“心理測定尺度集 ”, 東京都, 株式会社サイエンス社，2001.

[12] 橘木俊詔，日本の経済格差: 所得と資産から考える，第 590 巻，岩波書店，1998.

[13] 豊田敬，“不平等解析: ジニ係数と変動係数”，経営志林，vol.41，no.4，pp.131–135，2005.

[14] 中村和之，“経済指標の見方・使い方”，
http://www.pref.toyama.jp/sections/1015/ecm/back/2005apr/shihyo/.

[15] 北山聡，“組織内コミュニティの計量: ジニ係数とべき分布の視点から”, 東京経済大学コミュニケーション学会, コミュニケ-ション科学 029 号, pp.3–16, 2009.

[16] R.M. Dewan, M.L. Freimer, A. Seidmann, and J. Zhang, “Web portals: Evidence and analysis of media concentration”, Journal of Management Information Systems, vol.21, no.2, pp.181–199, 2004.

[17] 上田泰，組織行動研究の展開，白桃書房，2003.

[18] wikipedia.org,“Wiki”，
https://ja.wikipedia.org/wiki/ウィキ.

[19] 日本ムードル協会，“Moodle”，
http://moodlejapan.org/home/.

PBL におけるチーム比較を簡便化するための試行的取り組み

森本 千佳子
東京工業大学
morimoto@cs.titech.ac.jp

松尾谷 徹
デバッグ工学研究所
matsuodani@debugeng.com

要旨

本研究の目的は，プロジェクト型チーム学習の教育効果を，実務経験のない教員でもできるだけ容易に把握できるようにする点にある．その第一歩として，本稿ではチーム比較を簡便化する手法の試行結果について報告する．

ソフトウエア開発は複数のメンバーが集まったプロジェクト形態を取ることが多い．それを踏まえて，近年日本では情報系学生の教育にプロジェクト型のアクティブ・ラーニング手法である Project Based Learning （PBL）を取り入れた教育が増加している．PBL の狙いは，チームでの協働をシミュレーションすることを通して課題解決手法およびチームマネジメントを体験学習させることにある．しかし，従来の講義提供型の授業と異なり，チームをベースとした PBL ではその学習効果の把握手法は未だ確立されていない．とくに，実務経験のない教員にとっては，学習の肝となるチームの状態がどうなっているかを客観的かつタイムリーに把握することが難しい．

本研究では経済学の分野で所得格差の把握に用いられるローレンツ理論のジニ係数を PBL 授業への参加状態を把握するチーム貢献係数として応用し，チーム比較を試行した．その結果，チーム貢献係数によってチーム比較が簡便化できたことを報告する．

1. はじめに

組織経営の重要課題のひとつに、チームによる業務遂行の高品質化・効率化がある[1]．ソフトウエア開発は，複数のメンバーが集まったプロジェクト形態を取ることが多く，ソフトウエア産業においても，チームの生産性への関心は高い[2]．

しかし，プログラムを書くという行為はもともと高度に専門化した知的個人作業である．よって，ソフトウエア開発プロジェクトには高度専門職である個人を共通のゴールをもった集団（チーム）としてまとめ，成果を導く難しさがある．そこでチームをマネジメントする手法について，松尾谷らのパートナー満足研究（PS 研究）等，ソフトウエア開発のチームビルディングに関する様々な研究が行われている[3, 4, 5]．しかし，ソフトウエア開発プロジェクトの成功率は未だ約 3 割とも言われており[6]，引き続き研究と実践の蓄積が待たれている．

こうした産業界の背景から，情報系分野の高等教育に対し，実践的な高度 IT 人材育成の要請が高まっている．中でも，実践的な教育手法として Project Based Learning（PBL）が注目されている[7]．しかし，PBL 教育を行う教員に実際のソフトウエア開発プロジェクト経験が少なく[8]，特にプロジェクトマネジメントの面で指導が困難であることが指摘されている[9]．最近では実務でのプロジェクトマネジメントの経験が少なくても PBL のプロジェクト進捗を把握する方法として，GitHub のコミット数やチケット駆動開発のチケット情報等を活用する試みが行われている[10,11,12]．しかしこれらの取り組みは，データの蓄積や PBL 環境の構築に教員負担が大きい．

そこで本研究では，経済学の分野で所得格差の把握に用いられるローレンツ理論の「ジニ係数」を，PBL の取り組み状態を客観的に把握する手法として適用し，チーム状態を比較することを試みた．

本稿は以下の章で構成される．まず，次の 2 章で関連する先行研究と課題について整理し，3 章で本研究の課題と提案するアプローチについて述べる．4 章では実際の PBL での適用結果の分析と考察，5 章で全体のまとめと今後の適用可能性について述べる．

2. 関連研究

2.1. 情報系分野における実践教育

アメリカにおける実践教育

アメリカではまず 1950 年代に医療分野において，従来の「黒板を使った講義型授業」が現場のニーズに合っていないという医療界の要請から，実際の医療現場で起こっている問題を取り扱う Problem Based Learning が始まり，その流れはエンジニア教育に

も普及した[13]. Problem Based Learning では問題解決手法の習得に重点が置かれたが，エンジニアリング教育においては，チームで仕事を行うことが主流であることから，1960 年代以降，Problem Solving を包含した Project Based Learning (PBL)が教育に取り入れられ始めた[13]. その後，この動きはヨーロッパやオーストラリアにも拡大し，現代でも各大学で積極的に取り組まれている教授法の一つとなっている.

アメリカでは情報系を含む多くの大学のエンジニアリング教育に「キャップストーン・プログラム」が設置され，現在でも主に卒業研究として，企業の協力のもと，PBL によって実際の課題解決に取り組む教育が行われている[14]. キャップストーン・プログラムでは学習効果促進のために積極的な産学連携が行われ，企業からの派遣者も含めた Student Management Team(SMT)を作り，学生に積極的な関わりを行うことで，ひとりひとりの教育効果を確認，指導している[15]. 産業界からのキャップストーン・プログラムに対する評価は高いものの，課題の設定，授業環境の構築，SMT による教員の負荷等が大きく，効果的かつ効率的な授業運用方法が必要とされている[15].

日本における実践教育

日本では，中央教育審議会が 2012 年の中教審答申で大学等の高等教育にアクティブ・ラーニングの積極的な導入を推奨している[16]. アクティブ・ラーニングとは生徒・学生の積極的参加型授業の総称である[17]. この答申に先立ち，日本のエンジニアリング教育では，この数年，産業界からの強い要請を背景に，PBL 教育が盛んになっている[8]. 特に情報分野においては，2006 年から文部科学省の「先導的 IT 人材育成事業」が開始したが，そこには現場の課題に取り組む要請が盛り込まれている[18]. また，後続事業として 2013 年に開始した「実践的 IT 人材育成事業」では PBL を教育カリキュラムに必ず盛り込むこととされており，全国の情報系コースをもつ大学で PBL が盛んに行われている[19].

PBL の教育効果とチーム状態の把握

PBL には取り扱う内容（課題）に関する学習と，プロジェクトマネジメントに関する学習の 2 つの要素が含まれる. ソフトウエア開発 PBL ではソフトウエアエンジニアリングとプロジェクトマネジメントが学習対象となる. 教員が教育効果を把握するには，学生それぞれがどの程度，学習内容を理解しているのかを知る必要がある. しかし，チーム学習においては，学生個々の負担度のバラつきや，授業での学習時間以外でのチーム学習があるなど，教員が実際のチーム状態を把握するのは難しく，一般的に，最終成果発表会の発表内容や，テスト，レポートなどでチーム学習の成果を把握することが多い[17].

実務でのソフトウエア開発プロジェクト経験の少ない教員にとって，特にプロジェクトマネジメントを実践的に指導するのは非常に困難とされ，教育効果を上げるために産業界の企業人とのコラボレーションが推奨されている[15,20]. しかし，すべての PBL において企業からの支援が得られるわけではない. そこで実務経験が無くとも PBL の教育効果を上げるために様々な取り組みが報告されている.

例えば，チームに一歩深く入り込む方法として，PBL の途中で外部ファシリテータによる「質問会議」手法を用いた振り返りを行ったり[21]，PBL 専用の演習環境支援システムを提供したり[12,22]と，教員がそれぞれのチームに密接にコンタクトしながらチーム状態を把握する試みが行われている. しかし，これらの取り組みは実務家の助言が必要，独自のマネジメント環境を構築する負担が大きい，などが課題として挙げられている[12,21].

近年では，PBL の開発環境として GitHub 等のオープン・バージョン管理システムを使うことも増えている. そこで GitHub のコミット数やチケット駆動開発のチケット発行数を元にチーム進捗を把握する方法も提案されている[11,23]. また，プロジェクトデータベースを構築し管理データを格納，分析する試みも行われている[10]. これらの取り組みはチーム状態を把握するのに一定の成果を出しているものの，過去の取り組みデータの蓄積が必要であったり，全チームが使えるマネジメント環境やプロジェクトデータベースを準備する必要があったり，データの入力漏れチェックが必要であったりと，教員の負担は大きい. また，教員による管理のために，本来の PBL 作業以外の追加作業を学生に負担させることにもなっている. よって，学生負担を増やすことなく，教員が容易にチーム状態を把握する方法が模索されている.

2.2. 学習効果の把握

学習効果の把握手法としては「ルーブリック」が有名である. ルーブリックとは，レベルの目安を数段階に分けて記述して，達成度を判断する基準を示したもので，アメリカの大学教育で広く活用されている[24]. 近年，日本においても初等中等教育を中心に利用が進んでいる学習評価手法である[25]. ルーブリックに詳しい梶田は，テスト法・レポート法・観察法・作品法などさまざまな評価手段と，知識や技能など評価したい内容を組み合わせて評価設計することを推奨している[26]. しかし，日本の大学におけるアクティブ・ラーニングでの適用はその基準作りが進められている段階にある[25]. また教員側からはアクティブ・ラーニングの活用・評価に伴う負担増大を不安視する意見も多く，教育機関のあり方も含めた検討が必要とされている[27].

2.3. チームマネジメント研究

組織研究によると，チームとは，ある共通の目的の元に集まっ

た集団のことを指す[28]. チームのマネジメントについては，経営管理や組織行動，また，社会心理学など古くから様々な分野で研究が行われている．例えば経営学の視点では，太田は，個人の力を組織に活かすために従来の組織が個人を「抱え込む」マネジメントから個人が活躍できる「主体的チーム」マネジメントが 21 世紀の主流になるとしている[29]. また鈴木は，働く上で個人は組織ではなく仕事にコミットし，知的専門職は所属企業より自身が所属する専門チームに帰属性があるとしている[30]. 最近では，ビッグデータ視点での研究アプローチとして，デジタル社員証のデータを分析したビジネス顕微鏡による組織コミュニケーション研究も行われている[31].

ソフトウエア開発におけるチームビルディング研究では，チームの形成段階を「形成期・騒乱期・規範期・実行期・散会期」と定義したタックマンモデル[32]が有名である．榎田・松尾谷[5]はタックマンモデルを発展させ，チームビルディングの段階とパフォーマンスを関連づけた榎田・松尾谷モデルを発表し，産業界に広く支持されている．また，それを含めた一連のチームビルディング研究の成果[3, 4]では，チーム初期(形成期)での人間関係構築の重要性とチーム状況または形成段階に応じた適切なチームマネジメントの必要性が述べられている．また，チームの状態を，仲間意識や規範意識で判別する測定尺度の開発も行われている[3].

これら一連の研究は産業界で一定の評価を得，特にチームビルディング測定尺度はソフトウエア開発現場でも活用されているが[33]，教育現場での PBL にはほとんど活用されていない．その理由のひとつとして，教育カリキュラムがある．従来のソフトウエア教育ではチームビルディングを含んだグループ・ダイナミックス（集団力学）は，2008 年に J07-SE で学習項目として提案[34]されるまでカリキュラムに含まれていなかった．また，J07-SE 発表後も，大学の情報系教員では個人の研究活動が根底にあるため，チームでの協働経験が少なく，教員自身もチームビルディングを学ぶ機会が少ないためと考えられる．また，このことは教員がプロジェクトマネジメントを指導するのが困難とされることにも影響していると考えられる．

3. 本研究のアプローチ

3.1. 本研究の課題

前章より，本研究では，教員の負荷をできるだけ増やさず PBL の学習効果把握を可能にする手法の検討を課題とする．その第一歩として，本稿ではチームを簡便に比較し，特に手をかけるべきチームの検出を容易にする方法の探索に焦点をあてる．

3.2. ローレンツ理論

チーム状態を比較する手法として，経済分野で用いられるローレンツ理論の応用を提案する．ローレンツ理論は，ローレンツ曲線（Lorenz curve）とジニ係数（Gini inequality index, Gini ratio）を用いて，国全体の所得が各世帯にどのように分配されているのかを調べる時によく用いられる手法である[35].

ローレンツ曲線は，所得分配の傾向やマーケットにおける企業シェアなどの分析に利用されているもので，分布の集中度や不平等などの度合いをみる度数分布表から作成される．値の構成比の小さい順に並べて累積をプロットしたもので，全く偏りがなかった場合，ローレンツ曲線は対角線に等しい 45 度の右上がりの直線となる．偏りがあった場合，右下側にへこんだ曲線となる．ローレンツ曲線のカーブが大きいほど，偏りの程度が大きいことを示す．図 1 にローレンツ曲線の例を示す．例えば，所得の格差が広がり僅かな世帯が大きな所得を得ている不均等な状態の場合，ローレンツ曲線は右下に大きく歪む．

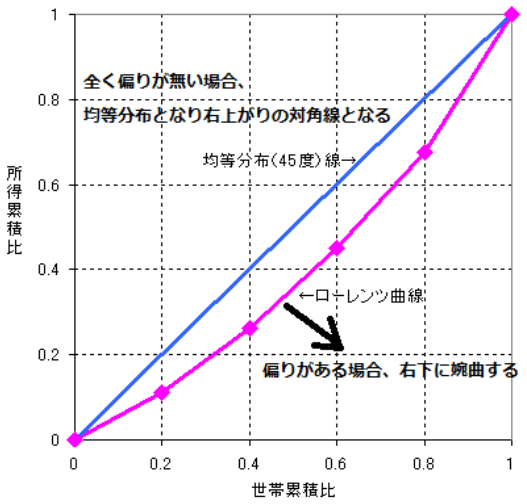


図 1 所得累積と世帯累積費を用いたローレンツ曲線の例

ジニ係数は，配分の不公平さを示す係数で，対角の 45 度線とローレンツ曲線で囲まれた弓型部分が，対角線の下半分の三角の面積に占める割合で表現され，0 と 1 の間を取る．配分が完全に均等の場合，弓型は 45 度の対角線と等しくなるため値は 0 となり，不均等になるにつれて弓型は膨らみジニ係数の値が大きくなる．

このローレンツ理論は，本来の目的である所得の不均衡を表す以外にも様々な分野で応用されている．例えば，ネットワークコミュニティの比較手法として，電子メールの発信件数をコミュニケーション量としてジニ係数を用いた研究[36]や，Web サイトの閲覧行動の偏りについてジニ係数を用いた研究がある[37].

3.3. ローレンツ理論を用いた PBL チームの比較

チーム状態を簡便に比較する方法の検討が本稿の目的である. その方法としてジニ係数を用いてチームの偏りを把握する. まず, PBL においてチームの何を比較するかを定める必要がある. ここでは, データ取得のために学生と教員負担をできるだけ増やさないという前提から, PBL の本来の目的において生み出すアウトプットを活用することを考える. 対象としては, PBL が取り扱う内容(課題)に関するアウトプットと, プロジェクトマネジメントの成果として生み出すアウトプット, 学習効果を把握するためのアウトプットがある. 本研究では, チームメンバー個々のアウトプット数が把握出来るものを用いて, チーム状態を比較する.

チームビルディング理論では, 最も高いパフォーマンスを出す「実行期」では, チーム全員がチームの目的達成に向けて協力的に取り組んでいる状態となる[5]. すなわち, 全員が等しくパフォーマンスを出している状態と言えよう. つまり, ローレンツ曲線が直線に近い状態(偏りが少ない状態)がチームビルディングできていると考えられる. そこでジニ係数の扱い方として, 偏りの大きさを表すジニ係数を逆に取った,「1-ジニ係数」を, チームビルディング状態を把握する係数として用いる. この係数を, 仮にチーム貢献係数(Team Contribution Ratio)と呼び, チーム貢献係数が高いほど, 全員がチーム活動に均等に取り組んでいると考える.

4. 提案手法の試行と評価

ここでは, 2014 年に実施した PBL のアウトプットを用いてチーム貢献係数を算出し, チーム状態の比較を行う. チーム貢献係数で得たチームの特徴と, 教員が行った観察法の記録を比較し検証する.

4.1. 対象 PBL の概要

対象 PBL の受講生は, 情報系の学生を中心としたエンジニアリング系の修士 1 年の学生 34 人で全員が男子学生である. 学生は A～E の 5 つのチームに分かれ, 4 月から 12 月までの 8 ヶ月間 PBL に取り組んだ.

指導は産業界での実務経験のある 2 名の教員が担当した. ひとりの教員はこの PBL に関わり 6 年目, もう一人は PBL 担当 2 年目の教員である. 両者とも 20 年以上のソフトウエア開発実務の経験がある. PBL では, ソフトウエア開発の超上流工程からリリースまでを体験する. 授業は週に 1 回(約 3 時間)行われた. 授業は前期と後期に分かれていたが, 学生は前期後期で同じチームに所属した. 教員は主に 4 月から 6 月にかけて, 要求分析やプロジェクトマネジメントに関するミニ講義を行ったが, 原則として, 5 月からチーム運営は学生に委ねられた. したがって具体的な開発ソフトウエアの設計や, プロジェクトマネジメント手法, 開発手法の選定も学生に任された. チームによってはプログラミングスキルの不均衡を埋めるため自主的な勉強会を開催した. 7 月の中間発表ですべてのチームが要求分析を終え, プロジェクト計画を立てていることが確認できていた. 表 1 に PBL のスケジュールと教員のチームへの関わり量を示す.

表 1 PBL 年間スケジュールと教員のチームへの関わり量

	スケジュール	教員の関わり量
4月	チーム分け, 全体講義	大
5月	要求分析	大
6月	要件定義, 計画	中
7月	仕様設計	小
8月	(夏休み／自主開発)	無し
9月	(夏休み／自主開発)	極小
10月	実装	一部に大
11月	実装, テスト	小
12月	振り返り, 最終発表会	中
1月	個人振り返り発表会	中

教員のチームへの関わり量が変動しているのは, チームビルディングの段階に沿っているためで, 重要なチーム初期に手厚く関与し, それ以降はチームの成長に任せているためである. ただし学生にとってハンドリングが難しい騒乱期については, 長期化・複雑化していないかチェックを行い, 規範期に移行できるようサポートを行うことになっている.

教員によるチーム状態の把握は観察法による定性分析が中心で, 主に授業時間の机間巡回と, 個人ブログを通じて行った. また月次の進捗報告内容, 報告態度から定性的にプロジェクトマネジメント状態をチェックした. 特に個人ブログは毎週学生が学習したことを投稿し, それに対し教員が適宜コメントする双方向のやり取りで, 学習状況の確認を行うのに活用した.

4.2. チームの特徴

表 2～表 4 に A～E チームへの教員のチーム評価コメントの抜粋を記載する. コメントは教員間のメールからの抜粋である.

表 2 教員チーム評価コメント(7 月末)

	7月(中間発表前)	評価
A	楽しそうだがまとまってるかな？教員の話は聞いてない？	○
B	飲み会無し, ミーティング沈黙の人が多い, まとまりに不安	△
C	勉強会活発, なんか楽しそう	○
D	留学生が多いけどそれなりに進んでる, 勉強会無し？	△
E	○○がリードしてる, 授業外で集まってるか不明	○

表 3 教員チーム評価コメント(10 月 2 週目)

	10月(後期2週目)	評価
A	分裂，継続する？でも飲み会は頻繁で仲いいのどうして？	×
B	リーダ来ない，1名脱退，仕様の大幅見直しとリスケ	×
C	どんどん進んで順調，ただしテスト計画まだ，いつ？	○
D	技術リーダが支えてる，優秀，役割分担できてる感じ	○
E	一番進捗が早い，ユーザの試行が計画されてる！	○

表 4 教員チーム評価コメント(2 月 PBL 終了後)

	2月(今年のPBL振り返り)
A	最後何とか形になった．結局いいチームだったの？スキルの問題？
B	キツかったけど，残った4人は結束してたんじゃないか
C	共通の趣味もあり，いいチームビルディングが出来てた．理想．
D	個人発表にビックリした．仲いいと思ってた．そんな割り切ってたとは
E	地味だけど良いチームだった．手がかからなかった．

前期授業の終了時点では，どのチームも例年と大差ないチーム状況であり，チーム A とチーム B にやや不安があったものの，例年発生する要求定義工程での騒乱期も比較的短く収束したように見えた．しかし後期授業が始まると，チーム A とチーム B に脱退するメンバーが複数発生し，チーム崩壊の危機を迎えた．定性観察結果をチームビルディング理論[3]に照らすと，夏休みに両チームとも再び騒乱期に陥り，チーム A は規範意識の不足，チーム B は仲間意識の不足で騒乱期から抜け出せなくなったように見えた．この状況を改善するため，教員の関わり量はチーム A とチーム B に集中した．教員とチームで議論した結果，開発範囲の縮小等の対策を行い，残ったメンバーで最終発表会までに実装を完成させた．

PBL の終盤では，授業を離脱したメンバーがいたものの，逆にそのことでチーム A やチーム B はグループ・ダイナミックスを学習できたように見えた．また他のチームはチームビルディングの段階を順調に移行，グループ・ダイナミックスを学習したように見えた．しかし，1 月の個人発表で，チーム D の複数のメンバーから，自チームは単位さえ取れればいい，という人が集まっており，仲間意識が出来ず，チームとしては一体感を感じられなかったという発表が複数人からなされた．これは，チームビルディング段階が「形成期」で止まっていたことを示唆している．チーム D の状態は担当教員の定性分析でも気づかなかった事象であり，観察法に重きを置いたチーム把握の難しさを再認識することとなった．

4.3. チーム貢献係数による分析

A～E のアウトプット情報をもとに，チーム貢献係数を算出し，ローレンツ曲線によってチームを比較する．PBL のアウトプットとしてはソースコードやドキュメント，GitHub 情報，バグ件数など様々なプロジェクト管理データがある．また，授業として作成したレポートや個人ブログがある．ここでは授業で要求したアウトプットは授業に積極的に参加したことを示す可能性が高いことと，個人ごとの件数が明確であるという理由から，個人ブログの合計投稿件数を用いる．件数集計の対象期間は 4 月から 11 月までである．表 5 に各チームのチーム貢献係数を記載し，図 2～6 にチームごとのローレンツ曲線を記載する．

表 5 各チームのチーム貢献係数(個人ブログ投稿件数合計)

	A	B	C	D	E
チーム貢献係数	0.82	0.57	0.87	0.47	0.72

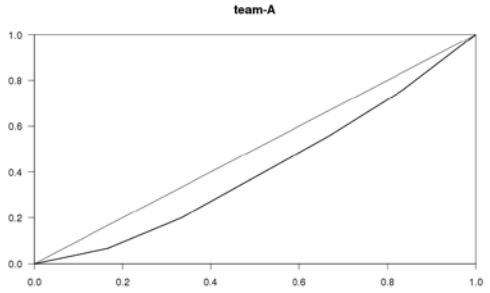


図 2 チーム A のローレンツ曲線

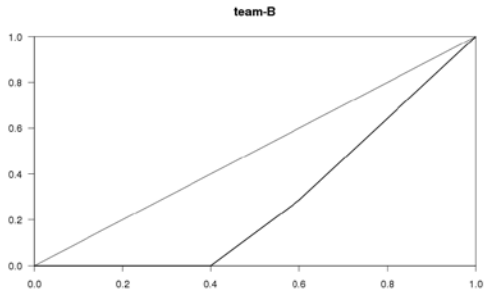


図 3 チーム B のローレンツ曲線

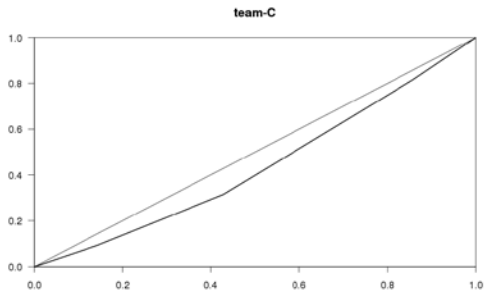


図 4 チーム C のローレンツ曲線

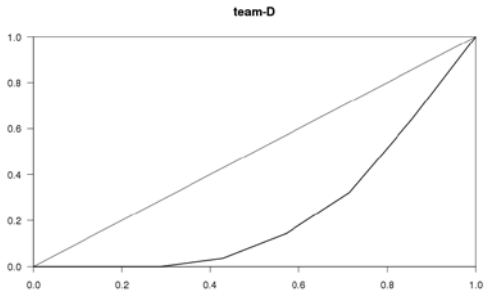


図 5 チーム D のローレンツ曲線

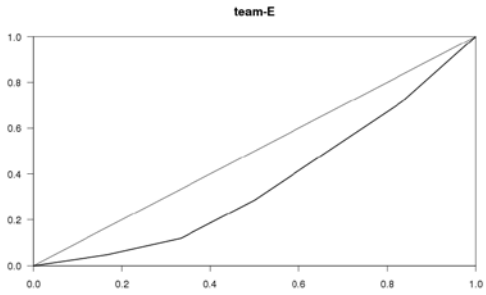


図 6 チーム E のローレンツ曲線

表 5 から分かるように，チーム貢献係数はチーム D が最も小さい．図 5 のローレンツ曲線も最も歪みが大きくなっている．また，チーム B は，まったくブログを書いていないメンバーが 2 名いた．この 2 名は後期から脱退したメンバーである．しかし図 3 の傾きをみると，書いているメンバーはほぼ等しい件数を書いていることが分かる．一方，最終発表会で優勝し，チームの自己評価も高かったチーム C はチーム貢献係数が大きく，偏りがあまり見られない．同様にチーム A もチーム貢献係数は大きい．

次に，経過把握の観点で，前期の成績からチーム貢献係数を算出する．前期の成績は，ブログ件数・ブログの内容の十分さ・レポートを元にどれぐらい授業に積極的に参加したかを教員が合議し，点数を出している．図 7 にチーム A～E のローレンツ曲線を示す．

図 7 を見ると，チーム D 以外はほぼ直線で偏りがない．全員がチームの中で均等な授業参加度合いだったと考えられる．チーム D だけは若干の偏りが見られる．

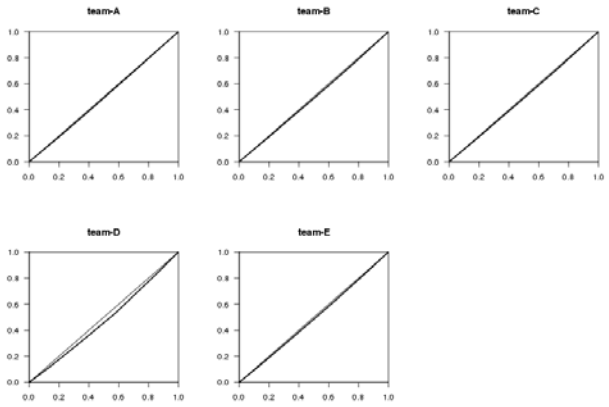


図 7 各チームの前期成績を用いたローレンツ曲線

4.4. 考察

チーム比較が出来たかどうか，特徴のあるチームを抽出できたか，という観点で考察する．まず，個人ブログ投稿件数と前期成績それぞれのローレンツ曲線を見ると，チーム D が他と違う様子を示していることが分かる．また図 3 から，チーム B が大きく 2 つの集団にわかれていることも推察できる．一方で，チーム A，チーム C はそれほど大きな差がない．チーム E も 5 チームの中では特徴的な傾向は見られない．これらのことから，チーム D は順調にチームビルディング段階を経ているのではなく，何かあるのではないか，と着目することが出来る．

観察法では問題があったチーム A だが，これについては，実際に脱退メンバーがいたことから，観察法から得た「ちょっと気になる」という気付きが正しかったと考えられる．しかし，PBL 授業以外では頻繁に飲み会を開催するなど，メンバー間の仲は良かった．この仲間意識がブログ投稿の行動パターンに影響したため，チーム貢献係数が大きくなった可能性がある．

チーム貢献係数は，あくまでその対象チームのアウトプットの偏りを示すものである．チームビルディングの測定尺度である規範意識や仲間意識は，高い方がチームとして高いパフォーマンスを示す[3]．高いパフォーマンスとは「実行期」に見られるように，チーム一丸となって全員が等しい積極さでその課題に取り組んでいると考えられるのではないか．よって，チーム貢献係数を用いることで，PBL への取り組みの偏りに気づけるのではないだろうか．すなわち，チーム貢献係数は，レポートや観察法など他のチーム状態の把握手法を補完することが出来るのではないかと考える．

5. まとめと今後の展望

本研究では，できるだけ教員の負荷を少なくした PBL の学習効果把握手法の検討を課題とし，その第一歩として，チームビルディングの観点から簡便にチームを比較する方法を探索した．具体的には，チーム貢献係数を用いてPBL への取り組みの差を比較した．その結果，本手法は注目すべきチームの発見に効果がありそうなことが分かった．チーム貢献係数は，チームのアウトプットから生じる主観に頼らない数値を使うことで，情報収集に特別な環境を必要としない簡便な方法といえる．また，単一の指標ではあるが，チームの違いをシンプルに表現できる点が使いやすいと言える．

しかし，チーム貢献係数と観察結果の比較のみで，チーム貢献係数がチーム状態を把握するのに有効だと断定することは出来ない．チーム貢献係数にどのアウトプット数値を用いればチームの何を比較できるのか，さらなる検証が必要である．ただ，実務経験のある教員による観察法でも気づけなかったチームの授業取り組み態度の差異に気づくキッカケを，チーム貢献係数によって得られた有用性は大きい．次の段階として，2016 年度の PBL で様々なアウトプットからチーム貢献係数を利用したチームの比較を行い，実績を蓄積したい．

謝辞

本研究は，野村マネジメント・スクールの助成を受けて実施した．

参考文献

[1] 山口裕幸,「エミネント・ホワイト（第5章チーム・マネジメント－機能的コラボレーションを創出する）」，小口孝司，今井芳昭，楠見孝(編著)，北大路書房, 2003.

[2] 徳丸宣穂,「日本のソフトウェア企業における経営管理 －技術選択および雇用・取引慣行との整合性」, NUCB Journal of Economics and Informations Science, Vol.53, No.2, 2009.

[3] 松尾谷徹,「IT に現場力は存在するのか:その計測と評価の試み」，ソフトウェア・シンポジウム 2014，ソフトウエア技術者協会, 2014.

[4] 池田浩,「チーム・メンタルモデルおよびチーム・パフォーマンスを規定する要因に関する検討 －チーム力およびチーム・リーダーシップの効果」，福岡大学人文論議，第 44 巻第 2 号,pp.293-309, 2012.

[5] 榎田由紀子，松尾谷徹,「Happiness ＆ Active チームを構築する実践的アプローチ ～チームビルディングスキルの開発～」，プロジェクトマネジメント学会誌, Vol.7, No.1, pp15-20, 2005.

[6] 日本情報システム・ユーザー協会,「ユーザー企業 ソフトウェアメトリックス調査 調査報告書」,2013.

[7] 文部科学省,「高度情報通信（IT）人材の育成に向けた文部科学省の基本戦略」, 2005.

[8] 九州大学，「情報工学系大学教員のための PBL 実践ガイド」，九州大学大学院システム情報科学府 情報知能工学専攻社会情報システム工学コース, 2012.

[9] 松澤芳昭，武田林太郎，大岩元,「学生主体のプロジェクトベース・ソフトウエア開発実践教育 －『教育的プロジェクトマネージャ』の導入と成果」，情報処理学会 コンピュータと教育研究会，情報教育シンポジウム論文集 2005，pp.37-42, 2005.

[10] 井垣宏，福安直樹，佐伯幸郎，柗本真佑，楠本真二,「アジャイルソフトウエア開発教育のためのチケットシステムを用いたプロジェクト定量的評価手法の提案」，情報処理学会論文誌, Vol.56, No.2, pp.701-713, 2015.

[11] 伊藤恵，木塚あゆみ，奥野拓，大場みち子,「過去の PLB の開発履歴を活用した PBL 運用支援」，日本ソフトウエア科学会第 31 回大会（2014 年度）講演論文集, 2014

[12] 西森年寿，加藤浩，望月俊男,八重樫文，久松慎一，尾澤重知,「高等教育におけるグループ課題探究型学習活動を支援するシステムの開発と実践」，日本教育工学会論文誌 29(3), pp289-297, 2005.

[13] Mills J, Treagust D, Engineering Education–Is Problem Based or Project Based Learning the Answer?, Australasian Journal of Engineering. Education, online publication (2003–04), 2003.

[14] Dutson A., Todd R., Magleby S. and Sorensen C., *A Review of Literature on Teaching Engineering Design Through Project-Oriented Capstone Courses*, Journal of Engineering Education, 86(1), pp.17–28, 1997.

[15] A.T. Chamillard, Kim A. Braun, *The Software Engineering Capstone: Structure and Tradeoffs*, SIGCSE '02 Proceedings of the 33rd SIGCSE technical symposium on Computer science education pp.227-231, 2002.

[16] 中央教育審議会，「新たな未来を築くための大学教育の質的転換に向けて～生涯学び続け、主体的に考える力を育成する大学へ～（答申）」，文部科学省，2012. http://www.mext.go.jp/b_menu/shingi/chukyo/chukyo0/toushin/1325047.htm (2016-03-02 参照)

[17] 山地広起，川越明日香，「国内大学におけるアクティブラー

ニングの組織的実践事例」, 長崎大学 大学教育機能開発センター紀要 3, pp.67-85, 2012.

[18] 文部科学省,「『先導的 IT スペシャリスト育成推進プログラム』の概要について」, https://www.ipa.go.jp/files/000024092.pdf , 2007. (2016-02-29 参照)

[19] enPiT,「分野・地域を超えた実践的情報教育協働ネットワーク」, http://www.enpit.jp/ ,(2016-02-29 参照)

[20] Yoshiaki Matsuzawa, Hajime Ohiwa, *A Model of Project-Based Learning to Develop Information Systems Engineers and Managers through "Collaborative Management" Approach (Advances in Learning Processes Chapter1)*, http://www.intechopen.com/books/advances-in-learning-processes, 2010. (2016-02-29 参照)

[21] 舘野泰一, 森永雄太,「産学連携型 PBL 授業における質問を活用した振り返り手法の検討」, 日本教育工学会論文誌, 39(Suppl.), pp.97-100, 2015

[22] 粂野文洋, 辻村泰寛, 大木幹雄, 山路秀美, 石原次郎, 松田洋,「地域組織連携による継続的なリアル PBL の試み －現状, 課題, 研究構想－」, 電子情報通信学会 信学技法, 2012

[23] 井垣宏, 柿本健,佐伯幸郎, 福安直樹, 川口真司, 早瀬泰裕, 崎山直洋, 楠本真二,「実践的ソフトウェア開発演習支援のためのグループ間比較にもとづくプロセスモニタリング環境」, 日本教育工学会論文誌 34(3), pp.289-298, 2010.

[24] 高浦勝義,「絶対評価とルーブリックの理論と実際」, 黎明書房, 2004.

[25] 沖裕貴,「大学におけるルーブリック評価導入の実際 －公平で客観的かつ厳格な成績評価を目指して－」, 立命館高等教育研究 14 号, pp.71-90, 2014

[26] 梶田叡一,「教育評価」, 有斐閣双書, 2005.

[27] 木村充, 山辺恵理子, 中原淳,「高等学校におけるアクティブラーニングの視点に立った参加型授業に関する実態調査: 第一次報告書.」, 東京大学−日本教育研究イノベーションセンター共同調査研究, http://manabilab.jp/wp/wp-content/uploads/2015/12/1streport.pdf , 2015.(2016-03-02 参照)

[28] Katzenbach JR, Smith DK , The Discipline of Teams, Harvard Business Review 71(2), pp.111-120, 1993

[29] 太田肇,「『個力』を活かせる組織」, 日本経済新聞社, 2000.

[30] 鈴木竜太,「組織と個人―キャリアの発達と組織コミットメントの変化」, 白桃書房, 2002.

[31] 辻聡美, 佐藤信夫, 紅山史子, 森脇紀彦, 矢野和男,「ビジネス顕微鏡による組織コミュニケーション改革の定量的評価」, 電子情報通信学会技術研究報告, Vol.111, No.308, SWIM2011-28, pp.59-64, 2011.

[32] Tuckman Bruce W., *Developmental sequence in small groups*, Psychological Bulletin, Vol 63(6), pp.384-399, 1965.

[33] 山川紘明,「職種を超えた連携におけるチームビルディング適用とその効果評価」, ソフトウェア・シンポジウム 2015, ソフトウェア技術者協会, 2015

[34] 阿草清滋, 西康晴, 沢田篤史, 鷲崎弘宣,「ソフトウエアエンジニアリング領域（J07-SE）」, 情報処理 Vol.49 No.7 特集情報専門学科カリキュラム標準 J07, pp.743-749, 2008.

[35] 中村和之, 園崎稔, 田平正典,「地方財源の負担と配分 －ローレンツ関数による接近－」, 日本地方財政学会編『地方分権と財政責任［日本地方財政学会研究叢書］』, 第 3 部, 勁草書房, 1999.

[36] 北山聡,「組織内コミュニティの計量 －ジニ係数とべき分布の視点から－」, コミュニケーション科学 29, 東京経済大学コミュニケーション学会, pp.3-16, 2009.

[37] Dewan R.M., Freimer M.L., Seidmann A., Zhang J.,Web Portals : Evidence and Analysis of Media Concentration, Journal of Management Information Systems, Volume 21, Issue 2, pp.181-199, 2004.

学生によるUX／アジャイルソフトウェア開発のYWTふりかえり事例

浦本 竜
北九州市立大学大学院
w5mcb002@eng.kitakyu-u.ac.jp

奥村 潤
北九州市立大学大学院
w5mcb003@eng.kitakyu-u.ac.jp

佐野 隼輔
北九州市立大学大学院
w5mcb004@eng.kitakyu-u.ac.jp

山崎 進
北九州市立大学
zacky@kitakyu-u.ac.jp

要旨

チーム開発の経験が少ない初心者のみで構成されたチームにおける開発は，様々な要因で失敗をすることがある．そこで，チーム開発の初心者であった我々が行った*3*件のアジャイル開発手法と*UX*を取り入れた開発経験を*YWT*でふりかえることにより，教訓を導く．導かれた教訓によって，初心者のみで構成されたチームにおける開発で，初心者にありがちな失敗を回避できるのではないかと考えた．そこで本稿では以下*2*点のことを目的としている．

- チーム開発の初心者に対して，失敗を軽減する為の教訓を導く
- *YWT*の使用感を考察する

*YWT*でふりかえりを行い，教訓を導くことができた．また，次の開発の方針を定めることができた．教訓を導くことができ，次の開発の方針が定めることができる*YWT*はふりかえり手法として有効であると考えられる．

将来課題として，導き出した教訓は，初心者のみで構成されたチームに適用して効果を確かめる必要がある．またふりかえり手法として*YWT*は普遍的に有効であるかを調査する必要がある．

1. はじめに

チーム開発の経験が少ない初心者のみで構成されたチームにおける開発は，様々な要因で失敗をすることがある．失敗の原因は，チームにおけるメンバー間の連携がうまく取れないことや経験不足等が考えられる．我々は2014年に行った開発において，初めてチームで開発する機会があった．その時の開発で多くの失敗を経験し，それらについての対策を考えた．また，我々は2015年においてもチームで開発する機会があり，2014年に考えた対策を実行してみたところ，2014年に比べ，良い開発になったと考えられる．そこで我々の開発事例をYWTというふりかえり手法を用いてふりかえり，教訓を導くことで，初心者のみで構成されたチームにおける開発が初心者にありがちな失敗を回避できるのではないかと考えた．そこで本稿では以下2点のことを目的としている．

- チーム開発の初心者に対して，失敗を軽減する為の教訓を導く
- YWTの使用感を考察する

本稿は以下の構成になっている．次の2章ではYWTについて紹介する．3章で3件の開発事例について紹介する．また，それぞれの開発事例についてYWTでふりかえり，失敗を軽減するための教訓を導く．4章でまとめる．

2. YWTとは

YWTというフォーマットを利用して，ふりかえりを行う．YWTとはやったこと（Y），わかったこと（W），次にやることの（T）の頭文字をつなげたものである．

まず，YWTについて，倉貫義人は次のように述べている[1]．

> 以前に紹介したKPTでふりかえるのが1～2週間程度だとしたら、YWTでは半年単位くらいでふりかえりと戦略について考えます。
>
> このYWTを考えるためには、将来に自分がどうありたいかといった漠然とでもビジョンがなければ難しいでしょう。YWTを考える機会は、自分の未来のことを考えるきっかけでもあるのです。

また，松尾睦は次のように述べている [2].

> 先日、ある方から「YWT」というふりかえりの手法を教えてもらった。(中略)「これは面白い」と思ったのだが、この手法は、コルブ（Kolb）の経験学習モデルに対応している点に気がついた。コルブによれば、人は「具体的な経験を→内省して→教訓を引き出し→新しい状況へ応用」することで学ぶという。つまり、YWT分析は
> 「何を実施したのか（具体的経験）」(Y)
> 「何がわかったのか（内省＆教訓）」(W)
> 「次に何をすべきか（新しい状況への応用）」(T)
> を考えることで、職場の経験学習を促進することができる。

文献 [1]，[2] より，3章で紹介する，我々のチームで行った3つの開発事例の開発期間が約4ヶ月であることとYWTでふりかえることによって，教訓を導き出し，初心者のみで構成されたチームにとって，より良い学習になるのではないかという2つの考えから，YWTを採用することにした．本稿では，我々が行った開発事例を以下のようにふりかえる．

1. やったこと（Y）で具体的体験をふりかえる
2. わかったこと（W）で1の内容から，わかったことや気づきをふりかえる
3. 次にやること（T）で2の内容から次にすることを考える

このフォーマットの，わかったこと（W）が教訓になり得る．改善が見込めるわかったこと（W）は次にやること（T）で，改善できるような仮説を立てる．その仮説を実際に次の開発で実行し，再びYWTのふりかえりを行う．YWTを繰り返すことにより，より良い学習につながり，より良い教訓が得られると考えられる [3].

3 開発事例とYWTによるふりかえり

本章では我々の3件の開発経験を説明し，YWTでふりかえりを行う．

3.1 2014年におけるWEBアプリケーションの開発事例

我々が2014年に行ったチーム開発によるソフトウェア開発事例について説明する．

3.1.1 開発概要

表1に開発概要について示す．

表 1. 開発概要

開発人数	5名
開発期間	9月20日～12月20日
アプリ名	バトリング
プラットフォーム	Ruby on Rails
ソースコード行数	約2500行 (自動生成の行も含む)

開発したソフトウェアの説明を以下に示す．

- 対象ユーザーは対戦型ゲームで遊ぶ人と定めた．
- 対象ユーザーが同じゲームで遊ぶユーザーと親交を深めることができるように，コミュニケーションをとれる機能を開発した．
- 実力を試したいユーザーがトーナメント式の大会を簡単に開催・参加できる機能を開発した．

3.1.2 ソフトウェア開発経験

ソフトウェア開発経験を2014年9月20日～10月31日，11月1日～11月30日，12月1日～20日の大きく3つの期間に分ける．

1. 2014年9月20日～10月31日について
   この期間では制作するソフトウェアの仕様とその対象ユーザーを定めた．10月20日までをRuby on

Rails の学習期間とした．学習期間が終了した後に開発を開始した．開発にはアジャイルソフトウェア開発を適用して行った．11 月の期間ではイテレーションを繰り返した．

2. 11 月 1 日〜11 月 30 日について
   - 進捗の確認の不十分で，プロジェクトが遅延した．従って，定例進捗会議を行うようにした．
   - プロジェクトを管理する役を決めていなかったため，プロジェクトの全体を把握することができなかった．従って，プロジェクトマネージャーを選出した．
   - 開発メンバー間でユーザーストーリーの優先度の認識がかみ合っていなかったため，ユーザーストーリーの優先度の統一を行った．
   - 設計の際，開発メンバー間で認識がかみ合っていないことが発覚したため，開発メンバーの認識を統一するために，プロトタイプとして，紙にインターフェースを手書きしたペーパープロトタイプを作成した．
   - 作成・編集されるファイルの変更履歴を管理していなかったため，全員でのファイルの共有とバックアップができなかった．また，ファイルの共有が出来ていなかったのでマージは変更したファイルを渡し，手動でマージした．従って，この問題を解決するために，バージョン管理ツールを導入してソースコードを開発メンバー間で共有した．
   - フィードバックを得ることなく開発を進めていたため，ユーザーに価値があるものを実装できているのか不明瞭だった．この問題を解決するために，想定ユーザーが使用している様子を観察することでフィードバックを得た．
3. 12 月 1 日〜26 日について
   チーム開発が終了した後，プロジェクト全体のふりかえりを行わなかった．チーム開発の経験者に指摘され，プロジェクトのふりかえりを行った．

3.1.3 やったこと（Y）

- 数々のチーム運営における失敗から，改善行動を行った．
  - 1 週間おきに定期会議を行って，全員がプロジェクトについて考える機会を設けた．
  - プロジェクトマネージャーの選出を行った．
  - ファイルの共有や履歴を管理するため，バージョン管理ツールを導入した．
  - 開発終了後に開発経験についてふりかえりを行った．
- UX を意識した設計を行った．
  - ペルソナ法を用いて，ユーザー像を定義した．
  - ペーパープロトタイプを作成した．

3.1.4 わかったこと（W）

- プロジェクトを管理していない時に比べ，プロジェクトを管理している時の方が作業が円滑に進む．
  - 定期会議を行うことにより，進捗の確認や現在の問題点等の情報をメンバー間で共有することで，次の週の開発における方針決めや，問題点に対応できる．
  - プロジェクトマネージャーがチームにいない時は，会議を行う間隔がばらばらであった．しかし，プロジェクトマネージャーがプロジェクトを管理することにより，1 週間おきの定期会議を行うようになった．それぞれのメンバーの進捗を全員で共有する場を設けることで透明性が向上し，プロジェクトが円滑に進む．
  - メンバー全員が常に最新のファイルを持つことができ，バージョン管理ツールを導入していなかった時と比べ，時間短縮となる．
  - メンバー全員参加のふりかえりを行うことで，それぞれのメンバーが感じた成功したことと失敗したことを共有でき，次の開発における仮説を立てることができる．
- UX を意識することによってプロダクトの品質が上がるだけでなく，一種の指標になる．
  - ペルソナ法を用いてユーザー像を定義したことにより，実装する機能がユーザー像に対してどのような効果をもたらすかという視点で考えることができる．

– ペーパープロトタイプを作成することにより，認識の統一を計ることができる．また，ペーパープロトタイプは，ページレイアウトを考える際に想定したユーザーが使いやすく，使用後の満足感が高くなるように意識して作成した．

3.1.5 次にやること（T）

- プロジェクト初期に，開発が円滑に進むような仕組みを導入する．
  – プロジェクトの状態を把握できるような仕組みを調べ，導入する．
  – プロジェクトマネージャーを選出する．
  – プロジェクトの始めにメンバー全員でプロジェクトについての考えを共有する．
  – バージョン管理ツールとしてGithubを用いたが，習得が不十分であったため，次のプロジェクトが始まる前までに習得しておく．
  – 定期的にふりかえりを行う．
- 開発を通して，UXを意識した開発の経験を積む．
  – ユーザーの価値を創造できるような方法を考え，導入する．

3.2 2015年におけるWEBアプリケーションの開発事例

我々が2015年に行った，チーム開発によるソフトウェア開発事例について説明する．

3.2.1 開発概要

表2に開発概要について示す．

表 2. 開発概要

開発人数	3名
開発期間	9月20日〜12月20日
アプリ名	Gift to
プラットフォーム	Ruby on Rails
ソースコード行数	約2600行 (自動生成の行も含む)

開発したソフトウェアの説明を以下に示す．

- 一言で言えば，店舗情報提供サービスである．これは現在地周辺にある店舗がわかり，店舗に行かせる動機付けをさせる狙いで開発した．
- 対象ユーザーは実店舗にプレゼントを買いにいく20歳前後の女子大学生と定めた．
- アイテム別のチェックボックスを用意し，検索機能とした．検索した結果，検索条件に合致したお店が表示される．さらに，お店を選ぶとお店の詳細情報を確認することができる．

3.2.2 ソフトウェア開発経験

ソフトウェア開発経験を2015年9月20日〜10月31日，11月1日〜11月30日，12月1日〜20日の大きく3つの期間に分ける．

1. 2015年9月20日〜10月31日について
   - チームビルディングの一環として，インセプションデッキ[4]を作成した．その際にプロジェクトマネージャーの選出を行った．使用するフレームワーク，チケット管理ツール，バージョン管理ツール等を決定した．
   - プロダクトの方針を決定するために，事前アンケートを実施した．アンケートの対象は20歳前後の大学生であり，100件以上回収した．
   - GitHubやRuby on Rails等の使用するツールについて事前に学習した．
   - ペーパープロトタイプを作成した．
   - ER図の作成を行った．
   - 我々で定めた，ユーザーが目的を達成できる最小限のプロトタイプを開発した．
2. 11月1日〜11月30日についてユーザーが目的を達成できる最小限のプロトタイプが出来た段階でアプリケーションを使ってもらい，その様子を観察すること，及び使用後の感想を聞くことという2点ををフィードバックとして開発を続けた．ユーザーテストは1度目に2人，2度目に3人の計2度，5人に対して行った．

3. 12 月 1 日〜26 日について
   ページデザインをブラッシュアップした．
   プロジェクト終了後に KPT で，プロジェクトのふりかえりを行った．

3.2.3 やったこと（Y）

- プロジェクトを始めるにあたって，プロジェクトが円滑に進むように新たな方法を導入した．
  - チケット管理ツールを用いて，プロジェクトのタスク管理を行い，タスクの可視化を行った．
  - プロジェクトの始めに，チームビルディングの一環として，インセプションデッキの作成を行った．この時，プロジェクトマネージャーの選出を行った．
- UX を意識したアプローチを新たに導入した．
  - アプリを作る方針を決定した後，想定ユーザーが実際どのような点で困っているのかを知るために，開発前にアンケート調査を行った．対象は 20 歳前後の大学生であり，100 件回収した
  - 想定ユーザーがある目的を達成できると予想される最小限の機能を開発できた段階で，実際に想定ユーザーに使ってもらった．使用中の様子を観察，また使用後にインタビューを行い，それをフィードバックとして次に開発する機能の参考にした．

3.2.4 わかったこと（W）

- プロジェクトの初期から開発環境を整える事で，余計なコストを抑える事ができる．
  - 2014 年の開発では，ホワイトボードに書き留めるという方法でタスクを管理していた．その時と比べると，残りのタスクの確認コストが低くなることや，タスクを優先度が高い順に並べているので取りかかるべきタスクが認知しやすく，タスクの分担も容易である等の利点があった．
  - 2014 年の開発では，認識のずれにより開発に余計なコストがかかってしまった．認識のずれはメンバー間におけるゴールの不一致や，大切な機能は何かということや，メンバーの役割は何かということ等で発生した．インセプションデッキでは，それぞれの役割，エレベーターピッチ作成等を定めるため，去年発生した余計なコストが比較的かからず，プロジェクトが円滑に進んだ．
- UX を高めるために行った方法はいずれも有効な手段であると考えられた．
  - 開発前にアンケート調査を行う事は，有効な手段であったと考えられる．しかし，アンケートの内容が深く練ったものでなかった為，得られた学びは少なかったと考えている．目的を明確にし，それに合わせたアンケート作りをする事で，得られる学びに違いがあるのではないかと考えた．
  - ユーザーテストにより，得られたフィードバックを元に，機能やタスクの優先順を選び開発する事で，次のユーザーテストの際，ユーザーからよいフィードバックを得ることができる．

3.2.5 次にやること（T）

- プロジェクト運営の観点で，チームの生産性を上げる．
  - チーム内で勉強会を開き，チームメンバーが互いに教え合う事で，チームのレベルが上がるのではないかと考えた．
  - 我々は開発におけるテスト工程の際，コードを記述し，実行画面を開き，手動で動作確認をするテストのみを行っていた．そこで，テスト駆動開発を試し，手動で動作確認をするテストと比較する．比較した結果を考察し，今後の開発に活かす．
- ユーザーの価値について考え，チームにおける価値も高める．
  - 目的をはっきりさせて，アンケートを作成すれば，さらに学びが大きくなるのではないかという仮説を立てた．次のプロジェクトでは，行動を起こす際，目的をはっきりさせる．

– プロジェクトの始めに定めたペルソナであるが，これが妥当なものであるかという事を第三者にレビューを行ってもらう事で，UX の高いプロダクトが作成できるのではないかと考えた．

3.3 IoT デバイスの開発経験

3.3.1 開発概要

表 3 に開発概要について示す．

本事例はインターンシップの一環として行った開発である．インターンシップ先の企業の方を顧客と見立てて，毎週ミーティングを行い，顧客の要求するシステムを開発した．開発したプロダクトは一言で言えば，作業管理デバイスである．複数の仕事を持っている人がそれぞれの仕事に費やした時間を記録できるような仕組みを Arduino を基調としたデバイスと kintone[5] というクラウドサービスで構築し，作業管理デバイスとしている．デバイスの部品として，主に Wi-Fi モジュール，タクトスイッチ，LED，SD カードリーダーを使用している．タクトスイッチを押すと，kintone と通信が行われ，タクトスイッチを押したときの時間を記録する．LED はデバイスの状態を表すインジケータとして使用した．SD カードリーダーは Wi-Fi の SSID やパスワードを SD カードに記録し，より利便性が高くなるように使用した．

3.3.2 2015 年における IoT デバイス開発経験

IoT デバイスの開発経験を 2015 年 10 月から 2016 年 1 月までを，1ヶ月ごとに分けて記す．

1. 2015 年 10 月 1 日〜10 月 31 日この期間は主に，仕様に関することを顧客と話し合ったり，技術調査をした．
   - 定期ミーティングの場で顧客の要求を調査し，要求を満たせるようなシステムを提案した．例えば，通信方式を Wi-Fi にすることや LED はデバイスの状態を表すインジケータとして使用すること等である．
   - 通信方式が決定してから，Wi-Fi モジュールの調査を行った．顧客の要求の中に HTTPS 通信をすることと可能な限り低コストなものを使用したいというものが存在したため，条件に合うような Wi-Fi モジュールを調査した．
   - 必要な電子パーツをリストアップして，調達した．
   - Wi-Fi モジュールを使うための初期設定を行った．
2. 2015 年 11 月 1 日〜11 月 30 日この期間は主に，デバイスを組み，通信テスト等を行った．
   - デバイスの回路図を作成し，回路図の通りにデバイスを作成した．
   - kintone 上にアプリケーションを開発した．このアプリケーションで作業時間の記録を確認することができる．
3. 2015 年 12 月 1 日〜12 月 25 日
   - タクトスイッチを押すと，時刻データを送信する機能を実装した
   - LED を実装し，デバイスの状態を知ることが可能となった．
   - SD カードモジュールをデバイスに組み込み，SSID やパスワードを登録を簡単にした．
4. 2016 年 1 月 5 日〜1 月 31 日この期間は最終調整を行った．
   - データが送信失敗した時に再送する機能を実装した．
   - 2016 年 1 月 26 日時点で，デバイスにかかったコスト計算書を作成した．

3.3.3 YWT による事例のふりかえり

3.3.4 やったこと（Y）

- クライアントの要求を調査し，システム案を作成した．
- デバイスに関するやったこと．
  – Wi-Fi モジュールの初期設定や必要な電子パーツのリストアップ等で時間がかかった．
- 文書を作成した．

表 3. 開発概要

開発人数	3 名
開発期間	2015 年 10 月 1 日〜2016 年 1 月 31 日
プロダクト名	作業管理デバイス
プラットフォーム	Arduino
ソースコード行数	660 行

– 消費電力に関する調査報告書を作成した.
– デバイスにかかったコスト計算書を作成した.

3.3.5 わかったこと（W）

- 顧客と認識をすり合わせる難しさと大切さがわかった.
- ハードの知識が乏しいということがわかった.
- 文書を作成する際，他者が読み，理解できるように書くことが難しいことがわかった.

3.3.6 次にやること（T）

- 顧客が要求の背景に考えていることを意識して，知る努力をする.
- デバイス構築のための電子パーツの知識が乏しいので，早急に必要になりそうなパーツをまとめて，先達のレビューを受ける.
- 普段からアウトプットする習慣をつけ，文章を書く練習をする.

4. まとめ

4.1 YWT によるふりかえりから得られた教訓

YWT によるふりかえりから以下 4 点の教訓が得られた.

- プロジェクトを管理していない時に比べ，プロジェクトを管理している時の方が作業が円滑に進む.
- UX を意識することによってプロダクトの品質が上がるだけでなく，追加しようとしている機能は想定ユーザーにとって価値があるのかといった考え方ができるようになるため，一種の指標になる.
- プロジェクトの初期からチームビルディングやタスク管理ツールを決定する等の開発環境を整える事で，認識のずれによる手戻りやツールを使わない事で発生する余計なコストを抑える事ができる.
- UX を高めるために，アンケートによる事前調査を行う事は有効である.

YWT で開発をふりかえる事によって，内省し，教訓を導く事ができた．この方法で得られた教訓は他の初心者のみで構成されたチームに適用し，有効性を確認する必要がある．また，YWT によるふりかえりの次にやること（T）から，2016 年における開発の方針が定まった．開発の方針を以下に記す.

- チーム全体の開発レベルを上げるために，チーム内で勉強会を開く.
- テスト駆動開発を試す.
- 行動を起こす際，目的をはっきりさせる.
- ペルソナ設定をより良いものにするために第三者にレビューをもらう.
- 顧客が要求の背景に考えていることを意識して，知る努力をする.
- デバイス構築のための電子パーツの知識が乏しいので，早急に必要になりそうなパーツをまとめて，先達のレビューを受ける.
- 普段からアウトプットする習慣をつけ，文章を書く練習をする.

4.2 YWT を使ってみてどうだったのか

3 章で紹介した開発事例では，開発を行っている最中は 1 週間ごとに KPT でふりかえり，また開発期間が約 4ヶ月の各事例について YWT で振り返った．YWT のわ

かったこと（W）はやったこと（Y）を振り返ったとき有効であった経験を教訓とし，さらに改善するための仮説があれば次にやること（T）で改善するための仮説を立てる．やったこと（Y）で改善する必要がある場合は次にやること（T）に改善するための仮説を立てる．このやったこと（Y）からわかったこと（W）を挙げる内省は KPT の K と P に相当すると考えられる．次にやること（T）はわかったこと（W）から挙げられることと同様に，KPT は K と P から T を挙げるので YWT の次にやること（T）と KPT の T は同質のものだと考えられる．YWT と KPT の違いは YWT のやったこと（Y）を可視化するところである．KPT の K と P は実経験から挙げていくとはいえ，プロジェクトの実体験を可視化しないため，YWT のやったこと（Y）を挙げてから，わかったこと（W）につなげるプロセスが KPT の K や P を挙げることより簡単だと考えられる．故に開発期間中をふりかえる場合は KPT が適しており，長期の開発経験をふりかえる場合は YWT が適していると考えられる．

また，やったこと（Y）で開発中に行ったことを列挙した．わかったこと（W）で具体的体験を内省し，教訓を導き出した．次にやること（T）では，開発活動を改善するための仮説を立てた．仮説を立てることで，次の開発の方針を定めることができた．教訓を導くことができ，次の開発の方針を定めることができる YWT はふりかえり手法として有効であると考えられる．

4.3 将来課題

導き出した教訓は，初心者のみで構成されたチームに適用して効果を確かめる必要がある．またふりかえり手法として YWT は普遍的に有効であるかを調査する必要がある．

5 謝辞

2014 年と 2015 年おける WEB アプリケーションの開発事例において，ハウインターナショナル　村上照明氏，IoT デバイスの開発経験において，名古屋大学　舘伸幸氏，AISIC　久米純矢氏にとくに謝意を記す．

参考文献

[1] 倉貫義人，ビジョンを叶えるために。個人でも出来る戦略を考える第 1 歩 〜 YWT を使った戦略の立てかたとは，http://kuranuki.sonicgarden.jp/2013/06/ywt.html

[2] 松尾睦，ラーニング・ラボ，http:///blog.goo.ne.jp/mmatu1964/e/585f7611fd43892f871e3aa0f2ffe4bb

[3] 松尾睦，職場が生きる人が育つ「経験学習」入門，ダイヤモンド社，2011

[4] Jonathan Rasmusson，アジャイルサムライ-達人開発者への道，オーム社 開発局，2011

[5] サイボウズ，kintone，https://kintone.cybozu.com/jp/

ソフトウェアの少子高齢化が急加速　～開発中心パラダイムに未来はあるか？～

増井　和也†

１．　はじめに

書名に「開発」を含むソフトウェア関連図書は多数出版[1]され，「ソフトウェア開発」という言葉に違和感を持つ人は少ないだろう。その図書群から，「開発」作業のイメージや期待値はソフトウェアを新規に創りだす創造性のある作業との印象を強く受ける。

では，ソフトウェアに関連する現場では，新規にソフトウェアを創作する作業が大半を占めているのだろうか。実際はそうではなく，長年稼働している既存ソフトウェアに対し，法律の改正，顧客や利用組織の要件変化，基盤ソフトウェアの更新，競合相手との優位性の拡大・維持，潜在不具合や稼働後の改修ミスの発見などに伴う修正または比較的小規模な機能追加（エンハンス）を行う作業が大半を占めるとの話を聞くことがある。このような作業の費用は当該ソフトウェアの稼働開始から 5 年間累計で初期開発費の 2 倍を超えるという報告[2]もあり，ソフトウェア技術者の多くがこのような稼働中ソフトウェアの延命作業に携わっているというのが筆者の認識である。既存ソフトウェアの延命作業が大半を占め，新規（初期）開発が極端に少ない状況が続くことを筆者は「ソフトウェアの少子高齢化」の状態と呼ぶことにする。

本稿は「ソフトウェアの少子高齢化」の状態が今後も拡大する根拠，従来の「開発中心パラダイム」が適用可能な範囲の縮小，将来に向けて既存ソフトウェアを中心にした新しい「保守中心のパラダイム」にシフトする必要性を，筆者個人の立場で論及する。

２．なぜソフトウェアは少子高齢化に向かうのか？

(1)筆者のソフトウェアに係る経験

筆者は大学で経済学を学んだ後，1976 年から（途中 40～50 歳台前半，管理職を兼務する時期もあったが）現在までほぼ一貫してソフトウェア技術者として従事してきている。その間，いわゆるプログラマ，SE，プロジェクトのリーダ(PL)，マネージャ(PM)，ソフトウェア現業部門の技術専門職，部門長の立場を経験した。また，OS・装置ドライバなどの基本ソフトウェアから各種業務・業種の受注アプリケーション，複数のパッケージソフトウェアまで，さまざまな種類やタイプのソフトウェアに関わってきた。

筆者の経験には，まったくのソフトウェア新規開発として各開発期間 1～２年の 6 プロジェクト（OS～受注アプリケーション，パッケージ）がある(含む PL,PM)。ただ，経歴の大半を占める作業分野は広義のソフトウェア保守[3),4)]であった。その中には，他者が開発し，長年の稼働中にさまざまな修正が無秩序に加えられてきたソフトウェアの保守を引取り，他者にいつでも引継げるよう保守体制を再構築して対応したものも多く含まれる。

筆者 30 歳台前半，UNIX の移植プロジェクトを(PL として)担当した。UNIX の処理理解に大量の C 言語プログラムを素早く読解する力が必要となり，プログラムの読解スキルを磨くことができた。筆者はもうすぐ 64 歳だが，この経験からか Java 等の他者開発のソースコードを読み解き，既定の保守プロセスの１タスクとして最適なソース修正を担うこともある現役の保守技術者である。

当該保守作業の依頼は今も減る気配がなく，フルタイム勤務を継続しつつ，長年の保守作業や保守組織運営の経験から得た総合的な技術や知見は，所属のソフトウェア・メインテナンス研究会の年次報告書，著作，セミナー等で発表している[4),5),6),7)]。

(2)ソフトウェア開発はますます縮小する

筆者の既報告に，稼働中ソフトウェアの価値について考察したものがある[8)]。そこでは，稼働中ソフトウェアの価値は障害等でそれが機能しなくなったことによる損失額（経済学でいう機会費用：**opportunity cost**）で評価すべきとした（初期開発費の高低は当該価値と無関係）。この見方から，再開発により現在の保守・運用費用を下げる投資についての損益分岐点（**break-even point**）の存在が予測できる。

一般業務システムの例であるが，仮に新開発技術により再開発が従来想定より大幅に低い費用で可能と見積もられたとしよう。だが，再開発に伴う導入側付帯費用として，要件定義時の関連部門対応，並行稼働（当該期間は二重の設備要），移行，各種運用マニュアル等再作成，再教育訓練，初期不良対応，　開発と並行稼働期間中の既存システム保守・運用（熟練技術者が再開発にシフトすれば，代替費用や障害拡大リスク高となる）などの費用が開発作業自体とは別に発生する。これら付帯費用は，再開発費の大小よりも業務影響範囲に大きく依存する。再開発版稼働後の保守・運用費の削減がある程度期待できた場合でも，付帯費用も含む全再開発費の回収に 10 年以上の期間が必要な投資計画となる場合も珍しくない。上記の再開発損益分岐点は開発中心指向の考え方を持つ人が期待するほど低くないと筆者は評価した。

筆者が大手 IT ベンダで（事業部組織だったが）技術担当の経営管理部門を兼務した際，半期，年度，中期(3 年)の期間予算達成が第一目標の状況では，各期間予算に影響が少ない費用で納まる保証がないかぎり，再開発の選択は困難ということも経験した。

反対に，筆者が考える既存ソフトウェアの価値モデルでは，保守や運用の重要性が再評価でき，当該価値の維持・向上を目的とした保守やシステム運用の適正費用がイメージできる。この見方では，今以上に保守や運用に費用を掛け，稼働中のソフトウェアの延命を図っても経済的または経営的に無駄な費用増とは言えないことになる（図 1[7)]）。

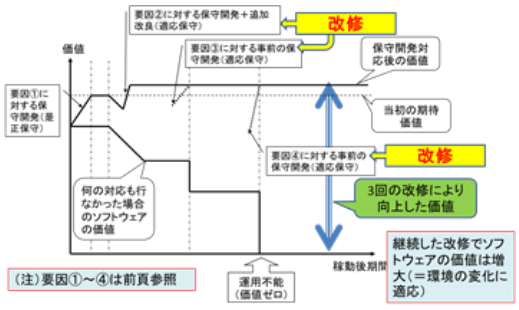


図 1 適切な保守対応後の稼働中ソフトウェアの価値変化

†SEA 正会員，　ソフトウェア・メインテナンス研究会（略称：SERC，　http://serc-j.jp/）　代表幹事

(3) ソフトウェア保守作業は隠されるが無くならない

ソフトウェア開発費が低いという甘言に乗り再開発してみたが，予算を大幅に超過し，保守や運用の費用も期待するほど下がらない場合がある。この場合，再開発投資回収が想定通りでない責任を少しでも逃れるため，実際の保守費用を別費目（例：次の開発費）で処理して保守作業が発生していないように見せる，本来対応すべき保守作業をせずに保守費用を下げた見返りとしてシステム障害リスクや障害対応の不手際リスクを高めてしまっている事例に，筆者は今まで複数遭遇している。

３．経営者の琴線に触れない開発中心パラダイム

「開発」に愛着を感じる人びとの中には，スマートフォンアプリや一部の新しい分野の組込ソフトウェアで，現在も大量の新規開発が行われていることに注目しているかもしれない。それらは学会発表，情報技術関連雑誌などの媒体で大きく取り上げられることが多いこともあるだろう。逆に取り上げられることが少ない既存ソフトウェアの少子高齢化対応が実際にはおおくあったとしても，一部の停滞した分野や再開発が必要な悪い状態の事例として紹介されることもある。

ただ，脚光を浴び，新しい分野で開発されたソフトウェアも，開発後はただちに既存ソフトウェアとなり，延命対象となるのである。新規開発の生産性・品質のさらなる向上が，実は保守されるべき既存ソフトウェアを急増させ，大量の高齢化ソフトウェアを残す方向に拍車をかける。

株主から期間損益の確保を徹底して求められる厳しい経営環境の中で，経営者，システムの所有者，利用者の意思として，既に稼働実績があり，安定していて高い価値を持つ既存ソフトウェアを（たとえそれが旧式のものであっても）延命を図る意思決定は今後も一般的になされていくものと筆者は経験から予測する。

長い目で見れば，開発は **discrete** であり，保守は **continuous** といえる。開発時点のみに着目する開発中心のパラダイムでは，長年稼働しているソフトウェアの価値を正当に評価はできず，システムのオーナである経営層に対し，「稼働中のソフトウェアは老朽化しているので新たに再開発が必要」というアピールは共鳴を得ることのないパラダイムとなってきていると筆者は分析する。

４．今高齢化ソフトウェアのケアが求められている

高齢化（老朽化）したソフトウェアが多数残存する状態を，「あるべき姿でない」と指摘するだけでは現状の解決にはならない。住む人の大半が高齢者の町に，大きな投資をして産婦人科の大医院を設立したとしてもビジネス的に失敗することが容易に想像できる。その失敗の原因は自分には無く，町の高齢化にあるのだとの主張に傾聴する人は少ないだろう。

世の中に既存ソフトウェアが大量に存在し，その寿命が延びている（高齢化している）現在，新規開発を想定した高生産性や高品質を求めるだけの技術的アプローチをいくら洗練し，研究し，また教育しても，それを活かせる場がますます少なくなっていく現状があることを正視すべき時にきている。

高齢化社会では老人をケアするためのグッズや医療サービスが必要となるように，高齢化した既存ソフトウェアをケアするサービス（保守サービス）がもっとも重要視される必要がある。

このような保守サービスを（既存ソフトウェアを残して）どのような技術で効率化し，高品質に対応していくか，高齢化ソフトウェアをケア（保守）する技術者が何を生きがいとして従事することができるか，今後のソフトウェアの分野の人々が自らで解決すべき大きな課題となるであろう。

その対応策として，あるべきソフトウェア保守プロセス実装には医療分野のプロセスをベンチマークし，実装していくことが有効[5]であると筆者は考えている。「保守は悪」という考え方を捨て，将来のためにソフトウェア保守を中心としたパラダイム（稼働中ソフトウェアが受診者，保守者が医師となるようなソフトウェア医療プロセスパラダイム）を早急に築き上げる必要性を提言したい。

５．まとめ

ソフトウェアの少子高齢化状態が急速に進む。その対応には従来の開発中心アプローチでなく，(いわゆる老朽化したものを含む)既存ソフトウェアを前提に，広い意味のケア（保守）を中心としたアプローチが必要である。

なお，「ソフトウェアの開発が無ければ保守もない」ということは明白な事実である。また，文献[3]のソフトウェア保守プロセスにおける「修正の実施」アクティビティからは，いわゆる「開発プロセス」を呼び出すというタスクが記されている。その点から，ソフトウェアの保守作業と開発作業には，引継ぎなどの連携すべき部分と実施上に共通なタスクも少なからず存在する。

ただ，従来は開発中心のプロセスを保守作業に無理に適用する方法が採られてきたが，ソフトウェアの少子高齢化状態では，保守プロセスから始める（付随する開発タスクも含めた）パラダイムへのシフトの是非について，今後の議論活性化を望みたい。

参考資料等

1) 国立国会図書館サーチ;16.2.19 時点，書名に「ソフトウェア開発」を含む 10～16 年出版の書籍検索ヒットは 734 件
2) ユーザ企業ソフトウェアメトリックス調査 2014　第 7 章　保守調査　分析結果　図表 7-71 保守費用分析
3) JIS X0161:2008 ソフトウェア技術－ソフトウェアライフサイクルプロセス－保守（ISO/IEC 14764:2006）で定義される是正保守，予防保守，適応保守，完全化保守のすべて
4) ～ISO14764 による～　ソフトウェア保守開発　増井和也，馬場辰男他　2007 ソフト・リサーチ・センター
5) ソフトウェア・メインナンス研究会　第 24 年次報告書 D グループ報告　個人研究　「ソフトウェア保守で何をすべきか医療プロセスから考える（増井和也）」
6) ソフトウェア・メインナンス研究会　第 15 年次報告書 D グループ報告　個人研究　「ソフトウェア保守の経済-Ⅰ（増井和也）」
7) 2015 年 2 月　「ソフトウェアの価値をさらに高めるために「保守・改修」で攻める！！」増井和也　日本情報システムユーザ協会オーブンセミナー
8) ソフトウェア・メインナンス研究会　第 18 年次報告書 D グループ報告　個人研究　「ソフトウェア保守の経済-Ⅳ（増井和也）」

非連続な実装を持つ連続値の境界値分析の考察

秋山 浩一
富士ゼロックス株式会社
Kouichi.Akiyama@fujixerox.co.jp

要旨

2016 年の閏日(20160229)は素数か否か？ という素朴な疑問から「ある年に素数はいくつあるか」という問題に発展し，この問題を解くプログラムが生まれた．ところがそのプログラムには境界値にまつわるバグがあった．本稿はそのバグの分析と対応について考察したものである．

年月日を“YYYYMMDD”の 8 ケタの数値で表現した場合その数値は必ず増加するが，月変わりや年変わりの境界では非連続となる．今回のバグは年変わりの境界で発生した．プログラムは容易にデバッグできたが，デバッグ後のプログラムが正しく動作することの検証について，計算量が多く答えがでなかった．冷静になって考え直したところ，数学的に解ける問題であることがわかった．

この経験を公開することで境界値にまつわるやっかいなバグの解決ヒントになることを期待している．

1. 背景

「閏日である2016年2月29日は素数か？」という問題が発せられた．年月日を「YYYYMMDD」の 8 ケタの数値で表現したときに「20160229」は素数か否かという問題である．素因数分解の簡単なプログラムを作成することですぐに「20160229＝929×21701」という解が得られるので「20160229 は素数ではない」という答えが得られた．

ここで，筆者は「年月日を同様の 8 ケタの整数で表現したときに2016年に素数はいくつ存在するか」という問題を考えてみた．簡単なプログラムを書き「素数は 19 個で内訳は 20160319, 20160401, 20160403, 20160529, 20160601, 20160607, 20160611, 20160709, 20160727, 20160809, 20160817, 20160821, 20160923, 20161007, 20161013, 20161019, 20161021, 20161027, 20161103 に違いない」という内容のツイートをしたところ知人でプログラムが趣味の居駒氏から，

```
ruby -rdate -rprime –e
'p    (Date.new(2016,1,1)..Date.new(2016,12,31))
.select{|date|date.strftime("%Y%2m%2d").to_i.prime?}.size'
```

という 1 ラインコードを DM で受け取った．その後，居駒氏から「上記は 140 文字に収まらないので」と，

```
ruby -rdate -rprime -e
'p (0..366).count{|i|(http://Date.new(2016,1,1)+i)
.strftime("%Y%m%d").to_i.prime?}'
```

という別解のツイートがあったが，改変後のプログラムには境界値にまつわるバグがあった．

2. 考察と解決策

2.1. バグの内容とデバッグ

バグはごく単純なもので「(0..366)」では 367 日チェックしてしまうという内容であった．「20170101」は素数ではないため，この境界値を含んでも含まなくても結果は 19 で改変前のプログラムの出力結果と変わらないという落とし穴もあった．

デバッグは「.」を一つ増やし「(0...366)」とすることで366を含まなくすればよい．

このとき，366を365に変えても同様の結果が得られるが，それではプログラミングの可読性(意図の獲得性)が悪くなるので，本ケースにおいては，Ruby の範囲式のうち終端を含まない「...」を使用すべきである．

2.2. デバッグ結果の検証

さて，デバッグ後の回帰テストについて居駒氏は「当該年の 1/1 と 12/31 に加えて，前年の 12/31 と翌年の 1/1 の 4 つのすべてが素数である年をテストする必要がある」と考えた．そこで，プログラムを組み，そのような年が存在するかのチェックを行った．それが見つかれば，その年で回帰テストを行えばよいからである．

ところが，2016 万をはるかに越えた 5 億年まで計算しても 3 つまで素数の年は数多くあるが 4 つ全てが素数の年は見つからない．その後，丸 1 日プログラムを走らせ続けても見つからなかった．(このときの検証環境は

OpenMP で 8 コア並列であった）

2.3. 問題の解決と考察

さて，プログラムにより4つの数字全てが素数の年がなかなか見つからないため，そもそもそのような年は無いのではないかと考えた．

存在することをプログラムで示すことはできるが，存在しないことをプログラムで証明することは難しい．たとえ，1兆年まで確認しても1兆1年で成立するかもしれないからである．（いわゆる「悪魔の証明」である）

そこで，プログラムではなく数学での証明を試みた．今回テストしたい数値は，前年の 12/31，当該年の 1/1 と 12/31，翌年の 1/1 の4つである．2016 年でいえば {20151231, 20160101, 20161231, 20170101}の 4 つの数字の全てが素数であるパターンである．

「これらの 4 つの数字が全て素数となることはない」ことについて，居駒氏により数学的に証明できることが分かった．

まず，この 4 つの数字を

（x1231，y0101，y1231，z0101 where x+1 = y = z-1）

と定義する．

素数は，5 以上では，6 で割った時の余りが1か5しか候補にならない．したがって x1231 が素数の場合，その余りは1か5である．

ここで，y0101-x1231 は，常に 8870 である．8870 を 6 で割った時の余りは 2 であるから，x1231 が余り 5 で，y0101 が余り 1 でないと両方素数にならない．

y0101 と y1231 の差は，1230 で，1230 を 6 で割った時の余りは 2 である．

したがって，x1231 と y0101 の両方が素数のとき，y1231 は必ず 6 で割ると 3 余りとなり，これは素数では無い．したがって，4 つの数字が全て素数であることはあり得ない．

さて，この経験からの学びをまとめよう．筆者は 3 つの学びがあったと考えている．一つ目は，日付が連続的に増えていくのに対して，年月日を“YYYYMMDD”の 8 ケタの数値で表現した場合の数値は非連続で増加するために実装上の境界が生じていたことである．このように仕様上あるいは日常的な感覚で連続性があっても実装上は飛び地があり，そこが境界になっている場合がある．ソフトウェアテストにおいてはまず，このような内部的に発生している境界に気がつく必要がある．そこに欠陥が生じやすいためである．

次に，閏年と言うことで 366 という値をプログラムに書き，範囲式を誤るという欠陥があった．これは，一つ注意したことで気持ちの中の警戒心が緩んだのであろう．「閏年についての考慮はしてある」という安心感が欠陥を誘発したのである．一つの落とし穴に対してプログラム上では複数の対応が必要な事がある．間違えやすいポイントは何度もチェックする必要があろう．

最後の学びは，一つの解決手段にこだわり過ぎると失敗するということである．数学的に証明しようと思えば 1 時間もかからずに済む問題が，プログラミングで答えを求めようとこだわると結果が得られなかった．

本件はシンプルな事例であるが，商品におけるやっかいなバグや，その解決方法の選択ミスで似たような経験を何度もしている．この事例から得た教訓を実業務で活かしていくことが重要であると考える．

対話型アプリケーションを対象とした 多種多様な環境に対応できるテスト実行自動化に関する手法

安達 悠　　岩田 真治　　丹野 治門
日本電信電話株式会社
adachi.yuu, iwata_shinji_s5, tanno.haruto@lab.ntt.co.jp

清水 誠介　　今井 勝俊
株式会社 NTT データ
shimizusi, imaikt@nttdata.co.jp

要旨

ソフトウェア開発において，1 つのテストスクリプトで様々なテスト実行環境に対応したテスト実行自動化を実現し，テスト実行自動化ツールの使い分けにより生ずるツール学習コストや，テスト実行環境ごとに生ずるテストスクリプト作成の手間を削減する手法を提案する

1. はじめに

現在，ソフトウェア開発ではテスト工程の工数削減のためにテスト実行自動化ツール（以降,「ツール」）が利用されており，様々な特徴をもったツールが存在している．しかし，テスト実行環境（端末種別や OS など）によってツールの使い分けが必要なため（例えば Selenium[1] はデスクトップ端末向け，Appium[2] はモバイル端末 (iOS, Android) 向け）ツールの学習コストがかかること（課題 1），また，レスポンシブデザインの Web ページのようにテスト対象が 1 つであっても，テスト実行環境ごとに異なるテストスクリプトを作成する必要があること（課題 2）が，テスト工程の工数削減の障壁となっている．

2. 課題解決アプローチ

前節で述べた課題を解決するために，1 つのテストスクリプトで多種多様なテスト実行環境に対応したテスト実行自動化を目指す．具体的には，1 つのテストスクリプトを各種ツールに対する入力形式に変換するテストスクリプト変換機能の実現を目指す.（図 1）

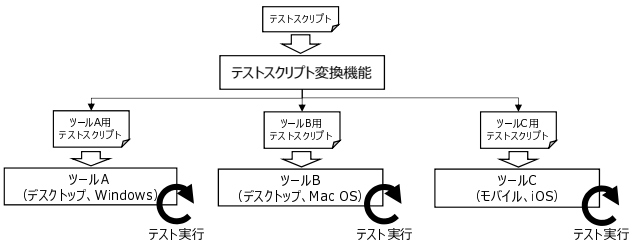


図 1. 課題解決イメージ

3. 課題解決の方法

前節で述べたテストスクリプト変換機能の実現にあたり，まず，ツール個別に提供されているテストスクリプト記述形式ではなく，共通的な記述形式を提供する．これにより，ツールの学習コストを削減する（課題 1 を解決）．次に，テスト実行環境に応じてツール個別に提供されているコマンドを 1 つに共通化して提供する．これにより，テスト実行環境ごとに発生するテストスクリプト作成の手間を削減する（課題 2 を解決）.

3.1. テストスクリプト記述形式の共通化

課題 1 の解決において，共通的なテストスクリプト記述形式を提供するにあたり，単に共通化するだけでなく，

表 1. キーワードによるテストスクリプトの記述例

Keyword	Object	Parameter
type	id=user	taro
type	id=passwd	1234
click	id=login	

より学習コストが低く，メンテナンス性の高いテストスクリプト記述形式を採用するべきである．現在利用されているツールのテストスクリプト記述形式は，ソースコードによる記述とキーワードによる記述に大別される．キーワードによる記述のほうが，コーディングスキルが不要であり，テストスクリプトの可読性も高いため，課題 1 の解決には，キーワードによるテストスクリプト記述形式での共通化が有効と考えられる．表 1 にキーワードによるテストスクリプトの記述例を示す．オープンソースのテスト自動化フレームワークである Open2Test[3] では，キーワードによる記述形式で共通化されたテストスクリプトによって，Selenium や Unified Functional Testing (UFT) [4] など各種ツールによるテスト実行が可能である．

3.2. コマンドの共通化

レスポンシブデザインの Web ページのようにテスト対象が 1 つであっても，テスト実行時には，端末種別によって操作が異なるため（例えば,「押下」の操作は，デスクトップ端末では「クリック」，モバイル端末では「タップ」），端末種別ごとにテストスクリプトを作成するのが現状である．課題 2 を解決するため，テスト実行環境に依存しない共通化されたコマンドを提供する．テストスクリプト変換機能でコマンドからテスト実行環境に対応した操作を解釈・変換し，各種ツールにて実行する．

図 2 にテストスクリプト変換機能の動作イメージを示す．予め，各種テスト実行環境に対するコマンド変換ルール（テスト実行環境に依存しない共通化されたコマンドからテスト実行環境に依存した実行形式のコマンドに変換するルール）をコマンド変換ルールデータベースに登録しておく（図 2（a））．テスト担当者は，共通化されたコマンドでテストスクリプトを作成し，実行先のテスト実行環境を指定してテストスクリプト変換機能へ入力する（図 2（b））．テストスクリプト変換機能は，コマンド変換ルールに基づいて共通化されたコマンドから実行形式のコマンドに変換および出力し（図 2（c）），指定したテスト実行環境でツールによってテストが自動実行される．

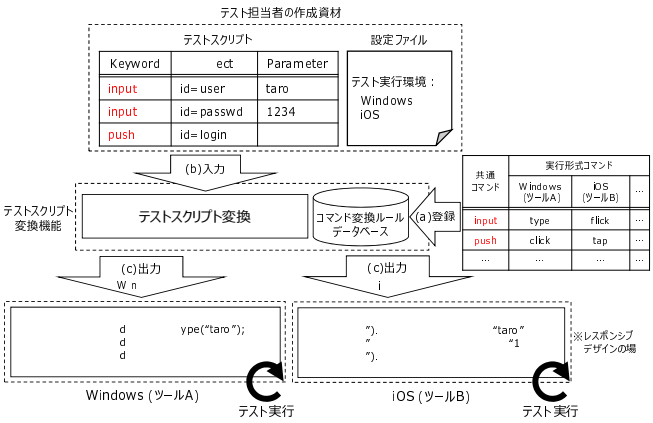


図 2. テストスクリプト変換機能の動作イメージ

4. おわりに

共通的なテストスクリプトの記述形式やコマンドを提供し，1 つのテストスクリプトで様々なテスト実行環境でのテスト実行自動化を可能とするテストスクリプト変換機能を提案した．現在，ソフトウェアテストの標準化の取り組みとして，ISO/IEC JTC1/SC7/WG26 ではキーワードによるテストスクリプト記述形式の共通化に向けた動きがあり [5]，本提案で提供するテストスクリプト記述形式を決めるにあたっての助力となっている．

今後，本提案の実現にあたり，テストを実行するツールごとのコマンド変換ルールおよびテストスクリプト変換機能の構築に向け，各種ツールベンダと連携していく必要がある．

参考文献

[1] http://seleniumhq.org/
[2] http://appium.io/
[3] http://www.open2test.org/
[4] http://www8.hp.com/jp/ja/software-solutions/unified-functional-automated-testing/
[5] ISO/IEC/IEEE DIS 29119-5, Software and system engineering –Software testing– Part5:Keyword-driven testing, 2015

非機能に関する要求仕様書作成の課題 ～開発現場における問題とUSDMを用いた解決方法～

水藤 倫彰
株式会社デンソー
TOMOAKI_SUITOU@denso.co.jp

古畑 慶次
株式会社デンソー技研センター
KEIJI_KOBATA@denso.co,jp

要旨

我々はLED駆動モジュールのミドルウェアを開発している. プロジェクトは既存のソースコードに対して新規の要求を実現し, 同時にリファクタリングを進める派生開発である. 開発では, 非機能要求についての手戻りが大きな問題となっていた.

非機能要求は, ソフトウェアの作り方に影響するため, 設計の再検討も必要となる場合もある. しかし, 開発現場では, 機能要求の実現を優先し, 非機能に関する要求仕様書の作成に時間を確保することは簡単ではない. また, その方法やプロセスも確立されていない.

本稿ではソフトウェアの作り方に対するレビュー指摘から, USDM と抽象化の概念, ISO25010 の品質特性を利用して非機能の要求仕様書を作成する方法を提案する.

1. はじめに

我々は車載メータに組み込むソフトウェアの LED 駆動モジュールのミドルウェアを開発している. 車載メータは製品のバリエーションが多く, 各バリエーションに対するソフトウェア要求も異なる. ミドルウェア開発は, 既存のソースコードに対して新規の要求を実現し, 同時にリファクタリングを進める派生開発である.

ミドルウェア開発は, 上位アプリケーションからの要求や実装するハードウェアに基づいて開発を進める. しかし, これらの要求は機能要求が中心で, ソフトウェアの作り方対する要求や品質要求などの非機能要求は, 要件定義の段階では文書化されていない. 実際の開発では, 非機能要求に対して下記で対応している.

(1) チェックシート
(2) 設計レビュー

(1)のチェックシートは, 過去に発生した問題から作成した確認項目集である. (2)の設計レビューでは, 有識者が成果物をチェックする.

上記の方法を通して得られる指摘事項に対応することにより品質を確保する. しかし, 現実には非機能要求について大きな手戻りが発生し, 開発プロジェクトが遅れる主要因となっていた.

本稿では, 開発現場における非機能要求の課題を整理し, USDM を用いた非機能要求の仕様化手法を提案する.

2. 非機能要求の課題

非機能要求は, ソフトウェアの作り方やアーキテクチャに大きく影響するため, 手戻りが発生するたびにソフトウェアを作り直す必要が出てくる. 特にテストフェーズで非機能要求のモレが発覚した場合は, 開発期間内にアーキテクチャの変更が困難な場合もある.

我々の開発のように要求仕様書に非機能要求が記述されていなければ, 設計レビュー, チェックシート等で問題をチェックする必要がある. しかし, レビューではレビューアの知見や技術力に依存し, 成果物が作成された後のチェックとなる. また, チェックシートでは, 対象となる製品ソフトウェアに対して一般的な記述となっているため, プロジェクトの特性に起因する要求に対応できない.

開発プロジェクトへの影響を考えれば, 非機能要求の文書化は必要不可欠である. しかし, ソフトウェア開発の現場では, 機能要求の実現を優先し, 非機能に関する要求仕様書の作成に時間を確保することは簡単ではない. また, 機能要求からソースコードを記述できるため, 納期を優先する組織では非機能要求の必要性を感じないという組織文化の問題もある.

開発現場では, 要件定義の段階で, 容易に, 時間をかけずプロジェクトの特性に合った非機能要求を仕様化する方法が必要である.

3. 現場における非機能要求の仕様化

現場における非機能要求の仕様化手法として,

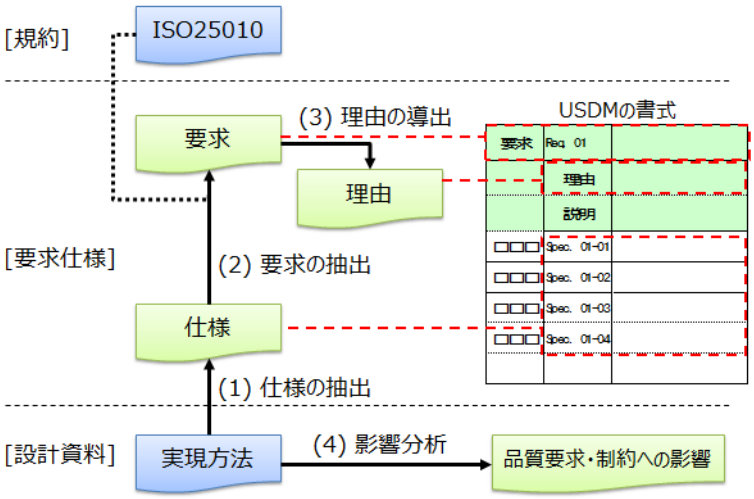


図1. 非機能の要求仕様の作成方法

USDM を用いた要求仕様作成方法を提案する.

現状の開発では, 非機能の要求仕様書をプロジェクトごとに新規に作成する時間は確保できない. 非機能要求は, 機能要求を満たす上での特性であるため, 同様の特性を有するプロジェクトの非機能要求を活用可能である. そこで, 非機能の要求仕様書を新規に作成するのではなく, 過去のプロジェクトから要求仕様を作成し, 新規プロジェクトに活用する.

要求仕様の作成にはUSDM[1]を利用する. USDM では, “要求”を「実現したいことを抽象的に表現したもの」と定義し, 実現範囲を示す. “仕様”は「“要求”に含まれる具体的な処理や振舞いを表現したもの」と定義している. これらは一般的な定義[2]とは異なるが, USDM では上記の定義により要求仕様を階層的に表現することができる. 階層構造で表現することにより, 要求仕様をまとめやすく, 新規プロジェクトでの活用も容易となる.

4. USDM を用いた仕様化手法

USDM を用いた非機能の要求仕様の作成方法を以下に説明する(図 1 参照).

本手法では, 実現方法としてレビュー指摘から, USDM の書式に基づいて非機能要求の“仕様”, “要求”, 理由の順に作成する.

(1) “仕様”の抽出

レビュー指摘への対応方法は, 具体的な設計, ソースコードの実現方法(どのように実現するか？)である. こうした実現方法のゴール(何を実現するのか？)を非機能要求の視点から検討することで, 実現すべき“仕様”を抽出する.

(2) “要求”の抽出

実現すべき“仕様” に対して抽象化を行い, その“仕様”を含む, 実現したい範囲を持った“要求”を抽出する. この時, 複数の“仕様”を包含する“要求”を求めるため, 一般的な品質特性である ISO25010[3]に対応付け, より広い実現範囲を持つ“要求”を導出する. 適切な特性がない場合は, 図 2 に示す非機能要求の分類[2]より, 法令遵守または制約に対応づける.

(3) 理由の導出

抽出した非機能要求が必要な理由を, USDM に従って導出する.

(4) 影響分析

レビュー指摘について, 他の非機能要求への影響を分析する.

本方法で作成した非機能の“要求”と“仕様”と共に, 具体的な設計, ソースコードの実現方法も活用できる可能性があるため, 実現方法を USDM の書式に追加する.

5. 実施結果

実際のプロジェクトにおける, 非機能要求に関するレビューの指摘事項 25 件から, 本手法を用いて非機能の要求仕様を USDM の形式で作成した. 作成した要求仕様では, “仕様”25 件, “要求”19 件となった.

事前に作成された非機能の要求仕様から, 要求仕様を選択するだけで, プロジェクトに合った非機能の要求仕様書を作成することが可能となった. また, “仕様”6 件が他と同一の“要求”に含まれたことから, 要求仕様の選

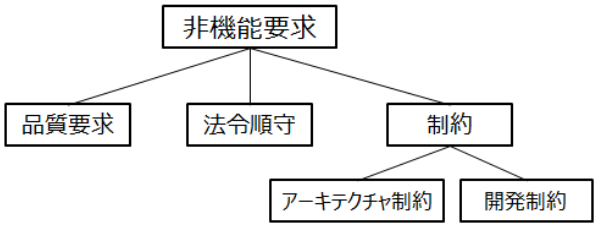


図 2. 非機能要求の分類

択時に，より少ない時間で判断をすることが可能である．しかし，レビュー指摘から要求仕様を作成しているため，レビュー指摘として出されなかった項目については要求仕様を作成できない．これは，本手法を多くのプロジェクトに適用し，要求仕様を増やすことで解決できる．

6. まとめ

現場において非機能要求を仕様化する時間の確保は難しい．要件定義の段階で，容易に，時間をかけずプロジェクトの特性に合った非機能要求を仕様化する方法が必要である．

本稿では，USDM の枠組みを使用し，ISO25010 の品質特性と抽象化の考え方を利用することで，レビュー指摘への対応方法から，非機能の要求仕様を作成する方法を提案した．

参考文献

[1] 清水吉男，要求を仕様化する技術・表現する技術 ~仕様が書けていますか？，技術評論社, 2005
[2] 情報サービス産業協会 REBOK 企画 WG 編集，要求工学知識体系 第 1 版,
[3] IPA SEC 監修，つながる世界のソフトウェア品質ガイド, IPA, 2015

ソフトウェア・シンポジウム 2016

～魅力的な「ソフトウェア開発」を目指して～

場所：米子コンベンションセンター BIG SHIP
主催：ソフトウェア技術者協会（SEA）

6/6（月曜日）	6/7（火曜日）	6/8（水曜日）
6/5（日曜日）併設イベント 『ものをつくる楽しさってなんだろう？』 [エントランスホール・1 階]	**9:00 ～ 10:00［各会議室］**※4 **論文・報告，Future Presentation** ※5 **10:15 ～ 11:15［各会議室］** **論文・報告，Future Presentation** ※5 **11:30 ～ 12:30［各会議室］** **論文・報告，Future Presentation** ※5	**9:00 ～ 12:00［各会議室］**※4 **ワーキンググループ・チュートリアル** ※6
12:30 ～ 受付［小ホール］※1	昼食	昼食
13:10 ～ 13:25［小ホール］ **オープニング** 実行委員長挨拶 宮田一平（アイエックスナレッジ） プログラム説明 小笠原秀人（東芝）	**14:00 ～ 18:00［各会議室］** **ワーキンググループ・チュートリアル** ※6	**13:30 ～ 14:00［小ホール］** **全体セッション** WG/TS からの報告と SS2016 のふりかえり 司会：小笠原秀人（東芝）
13:25 ～ 15:10［小ホール］ **オープニングキーノート** 『オントロジーとモデル理論に基づくソフトウェア開発の品質向上の実践 －上流工程支援と下流工程技術の両立を目指して－』 国立研究開発法人 産業技術総合研究所 情報技術研究部門 ソフトウェアアナリティクス研究グループ 上級主任研究員 和泉 憲明 氏 司会：宮田一平（アイエックスナレッジ）		**14:15 ～ 15:15［小ホール］** **クロージングキーノート** 『大山とともに生きる ～マインドマーク大山の１３００年物語～』 NPO 法人大山中海観光推進機構 （通称：NPO 大山王国） 理事長 石村 隆男 氏 司会：松本健一（奈良先端科学技術大学院大学）
15:40 ～ 17:40 ※2 **フォーラムセッション（1）［第 6 会議室］** ポスト・モバイル社会 －セカンドオフラインの世界－ **フォーラムセッション（2）［小ホール］** 地域貢献と研究成果を両立するソフトウェア工学教育の実践 －不十分な条件下の SE 教育の創意工夫－		**15:15 ～ 15:30［小ホール］** **クロージング** 表彰 大平雅雄（和歌山大学） 実行委員長挨拶 松本健一（奈良先端科学技術大学院大学）
18:00 ～ 20:00［国際会議室］※3 **情報交換会** 司会：宮田一平（アイエックスナレッジ）		

※1：オープニング以降の受付場所は以下になります．誰もいない場合は，070-6429-0240 にお電話ください（この電話番号は，会期中にしかつながりません）．
1 日目：「小ホール」，17:40 以降「国際会議室」/ 2 日目：午前「第 5 会議室」，午後「第 6 会議室」/ 3 日目：午前「第 6 会議室」，午後「小ホール」
※2：フォーラムセッション 1 と 2 は並行して行います．どちらに参加していただいても構いません．内容の詳細については p.2 をご覧ください．
※3：情報交換会は 17:40 に開場し，18:00 には“開始”します．お早めにおいでください．
※4：2 日目，3 日目はともに，8:50 に開場します．
※5：論文・報告発表と Future Presentation は並行して行います．どちらに参加していただいても構いません．会場については p.2 をご覧ください．
※6：時間は会議室が利用できる時間帯を示しています．各 WG の開催会議室については p.3 をご覧ください．
運営方法は WG ごとに異なります．詳細は各 WG のリーダまでお問い合わせください．
WG のリーダは，火曜日は 13:40 以降に，水曜日は 8:40 以降に，鍵を「第 6 会議室」まで取りにきてください．
終了後は設備を元どおりにし，水曜日の 12:30 までに，鍵を「第 6 会議室」に持ってきてください．

フォーラムセッション 1：6/6（月曜日）15:40～17:40 [第 6 会議室]

タイトル: ポスト･モバイル社会 －セカンドオフラインの世界－

講演者: 関西大学 社会学部 メディア専攻 富田 英典 教授

概要: 私たちは，モバイルメディアを利用し，その場で必要な情報をすぐにインターネットで調べたり，いつでも友人と連絡をとったりしている．つまり，「オフライン」において「オンライン」情報を常時参照しているのである．このような状況を「セカンドオフライン」と名づけたい．現代社会は，もはやモバイルメディア社会という範疇を超えた新しいもうひとつの社会へと移行しつつある．ここでは「可愛さ」の変遷など具体的な事例を取り上げながらその姿に迫る．

司会：鰺坂恒夫（和歌山大学）

フォーラムセッション 2：6/6（月曜日）15:40～17:40 [小ホール]

タイトル: 地域貢献と研究成果を両立するソフトウェア工学教育の実践 －不十分な条件下の SE 教育の創意工夫－

講演者: 阪南大学 経営情報学部 経営情報学科 大学院 企業情報研究科 花川 典子 教授

概要: 大学教育では学生教育の充実のみならず，地域社会貢献や就職支援，産学連携など多くを要求され，同時に自身の専門分野の研究成果も要求される．時間制約や人材制約の多い中，最大限の教育効果や研究成果を実現するための企業にも共通する創意工夫を紹介する．

司会：栗田太郎（ソニー、フェリカネットワークス）

論文･報告，Future Presentation

時間	第 2 会議室	第 3 会議室	第 4 会議室	第 5 会議室
9:00～10:00	形式手法／OSS	テスト	保守／レビュー	Future Presentation (1)
10:15～11:15	OSS／見積り／テキストマイニング	形式手法／保証ケース	要求獲得／保守	Future Presentation (2),(3)
11:30～12:30	プロセス改善／テスト	開発プロセス	チーム／ふりかえり	Future Presentation (3),(4)

◆6/7（火曜日）9:00～10:00

<形式手法／OSS> [第 2 会議室] 司会：鈴木正人（北陸先端科学技術大学院大学）

- 研究論文：VDM-SL 仕様からの Smalltalk プログラムの自動生成：小田朋宏（SRA）
- 研究論文：重大不具合に対する OSS 開発者の認識調査：松野祐介（和歌山大学）

<テスト> [第 3 会議室] 司会：秋山浩一（富士ゼロックス）

- 経験論文：大量の状態とイベントを持つプログラムの自動解析とテスト ～ 想定外のイベントを自動テストする ～：下村翔（デンソー）
- 経験論文：Flash メモリ管理における Concolic Testing の活用～メモリパターンを含む自動回帰テスト～：西村隆（デンソークリエイト）

<保守／レビュー> [第 4 会議室] 司会：栗田太郎（ソニー、フェリカネットワークス）

- 経験論文：性能トラブル解決の勘所：鈴木勝彦（ソフトウェア・メインテナンス研究会／株式会社日立ソリューションズ）
- 事例報告：レビュー要求分析・設計・実装試行でわかったこと：安達賢二（HBA）

<Future Presentation (1) > [第 5 会議室] 司会：中谷多哉子（放送大学）

- Future Presentation (1)：ソフトウェアの少子高齢化が急加速 ～開発中心パラダイムに未来はあるか？～：
  増井和也（ソフトウェア・メインテナンス研究会）

◆6/7（火曜日）10:15～11:15

<OSS／見積り> [第 2 会議室] 司会：鈴木裕信（鈴木裕信事務所）

- 研究論文：製品開発における OSS 導入のための OSS 事前評価手法確立に向けた調査：松本卓大（九州大学大学院）
- 研究論文：ISBSG データを用いた見積もり研究に対する IPA/SEC データを用いた追試：山田悠斗（大阪大学）
- 経験論文：週報のテキストマイニングによるリスク対応キーワード抽出：野々村琢人（東芝）

<形式手法／保証ケース> [第 3 会議室] 司会：佐原伸（法政大学）

- 事例報告：保証ケース作成支援方式について：山本修一郎（名古屋大学）
- 事例報告：VDM++仕様から C#コードを生成するツールの開発と評価：千坂優佑（仙台高等専門学校）
- 研究論文：システム理論的因果律モデルに基づくプロセス改善分析：日下部茂（長崎県立大学）

<要求獲得／保守> [第 4 会議室] 司会：小笠原秀人（東芝）

- 経験論文：ソフトウェア開発現場での Minute Paper の適用：岡本克也（エヌ・エム・エス）
- 事例報告：「保守あるある診断ツール」による保守課題の可視化事例：室谷隆（TIS）
- 研究論文：記憶力に基づくプログラム理解容易性評価尺度の追加実験：村上優佳紗（近畿大学大学院）

<Future Presentation (2),(3) > [第 5 会議室] 司会：植月啓次（パナソニック）／西康晴（電気通信大学）

- Future Presentation (2)：非連続な実装を持つ連続値の境界値分析の考察：秋山浩一（富士ゼロックス）
- Future Presentation (3)：対話型アプリケーションを対象とした多種多様な環境に対応できるテスト実行自動化に関する手法（前半）：安達悠（日本電信電話）

◆6/7（火曜日）11:30～12:30

<プロセス改善／テスト> [第 2 会議室] 司会：田中康（東京工業大学）

- 経験論文：開発の"待ち"，"遅れ"，"欠陥"を防ぐプロセス改善 ～PM 改善ナビを利用した"待ち"，"遅れ"，"欠陥"の改善活動～：比嘉定彦（アドバンテスト）
- 経験論文：ゲーミフィケーションを用いた探索的テストの効果報告：根本紀之（東京エレクトロン）
- 経験論文：ソフトウェア開発プロセスの違いによるテストプロセス成熟度の比較・考察：河野哲也（日立製作所）

<開発プロセス> [第 3 会議室] 司会：阪井誠（SRA）

- 事例報告：新商品分野でのソフト品質保証プロセスの構築事例：久木達也（リコー）
- 経験論文：情報システム開発におけるソフトウェア資産の上流シフトへの対応：松山浩士（サイバーリンクス）
- 経験論文：BRMS を適用するためのフィージビリティ・スタディ：李東昇（富士通システムズ・イースト）

<チーム／ふりかえり> [第 4 会議室] 司会：大平雅雄（和歌山大学）

- 経験論文：チームの協働状態を測る：Team Contribution Ratio ～手間のかかる質問紙からの脱却～：増田礼子（フェリカネットワークス）
- 経験論文：PBL におけるチーム比較を簡便化するための試行的取り組み：森本千佳子（東京工業大学）
- 経験論文：学生による UX／アジャイルソフトウェア開発の YWT ふりかえり事例：浦本竜（北九州市立大学大学院）

<Future Presentation (3) ,(4) > [第 5 会議室] 司会：西康晴（電気通信大学）／八木将計（日立製作所）

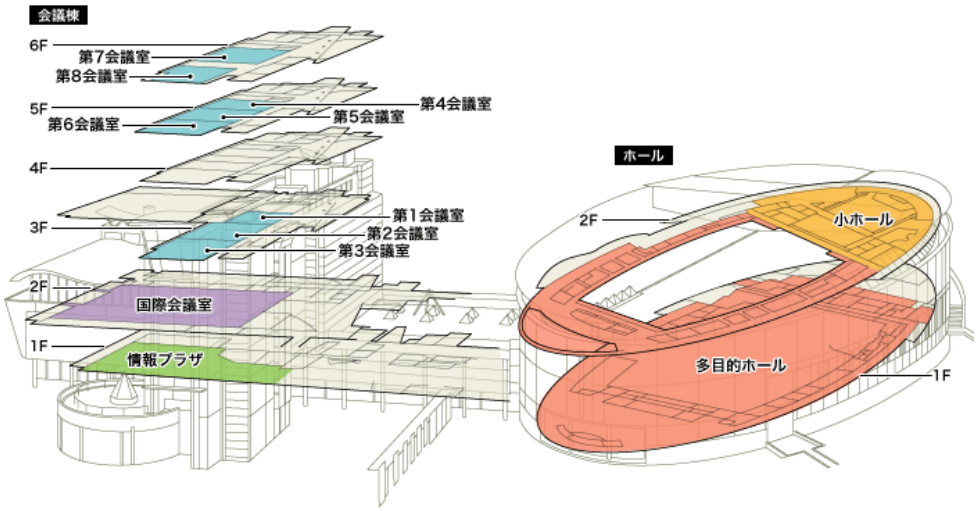
- Future Presentation (3)：対話型アプリケーションを対象とした多種多様な環境に対応できるテスト実行自動化に関する手法（後半）：安達悠（日本電信電話）
- Future Presentation (4)：非機能に関する要求仕様書作成の課題 ～開発現場における問題と USDM を用いた解決方法～：水藤倫彰（デンソー）

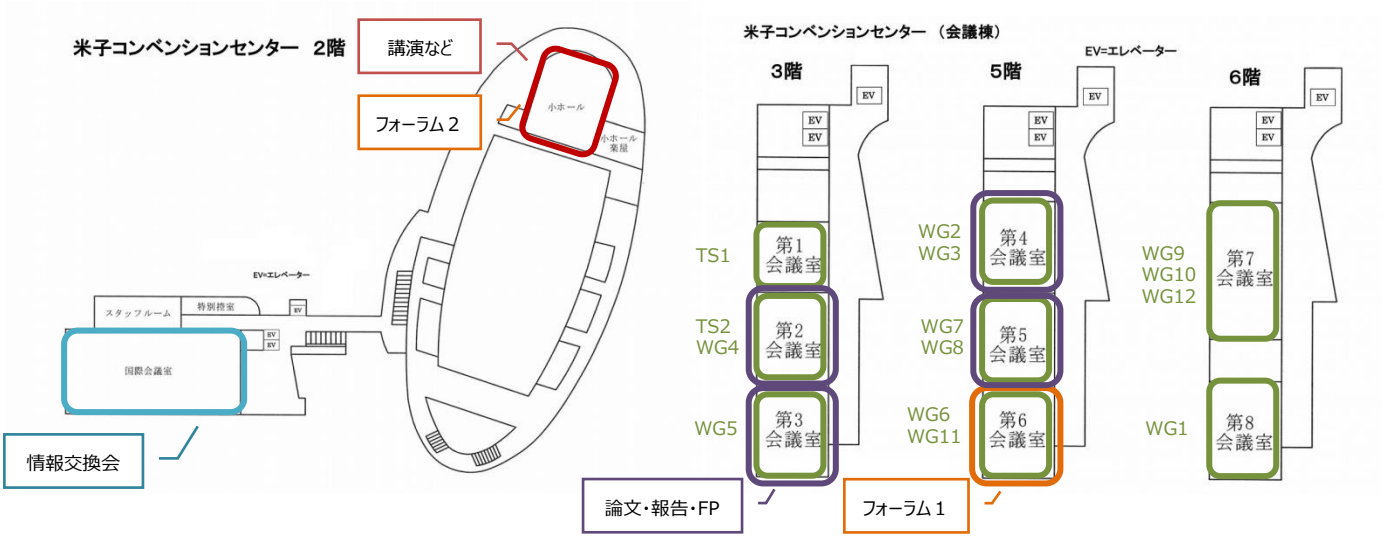
ワーキンググループ・チュートリアル 6/7（火曜日）14:00～18:00，6/8（水曜日）9:00～12:00

WG	略称	名前	場所
1	ED	自律的な成長力の育成	第 8 会議室
2	RV	リスクアセスメントをベースとしたソフトウェアレビュー	第 4 会議室 (1)
3	MT	不立文字から形式知に～ソフトウェア保守技術の伝承考える～	第 4 会議室 (2)
4	TA	チーム活動を外から比較する ～チーム協働状態を示すチーム貢献係数～	第 2 会議室 (2)
5	TT	状態遷移を含む統合テストの自動生成 <Concolic testing の進撃>	第 3 会議室
6	SS	ソフトウェアと社会	第 6 会議室
7	RQ	要求技術者の責任と社会システムの狭間	第 5 会議室 (1)
8	FM	Pre-formal ツールを用いた仕様記述の実践と応用	第 5 会議室 (2)
9	RC	アジャイル RCA＋SaPID による障害分析から対策策定までの一気通貫プロセスの体験	第 7 会議室 (1)
10	SD	サイズ・・見積もってますか？ - USDM で仮説見積もりと高精度のサイズ見積に挑戦しよう	第 7 会議室 (2)
11	SE	ソフトウェア開発の現状と今後の発展に向けたディスカッション	第 6 会議室
12	DC	D-Case	第 7 会議室 (3)

TS	略称	名前	場所
1	FC	ソフトウェア開発チームのためのファシリテーション入門	第 1 会議室
2	PT	パターン・ランゲージの再入門 再利用からクオリティの生成へ	第 2 会議室 (1)

【会場案内】 米子コンベンションセンター BIG SHIP





最新情報について

SS2016 の最新情報は，随時 Web ページに掲載いたします．公式ページの「新着情報」をご覧ください．

http://sea.jp/ss2016/news.html

Memo

ソフトウェア技術者協会（SEA）

http://sea.jp/ss2015/

6月に米子で「魅力的なソフトウェア開発」について話しませんか

ソフトウェア・シンポジウム 2016（SS2016 in 米子）

第 36 回目を迎える 2016 年のソフトウェア・シンポジウムでは，昨年から始めたフォーラムセッションとFuture Presentation という発表形式をさらに発展させたものにしたいと考えています．このほか，SS2015 に引き続き，論文発表や事例報告と，ワーキンググループで議論を行います．

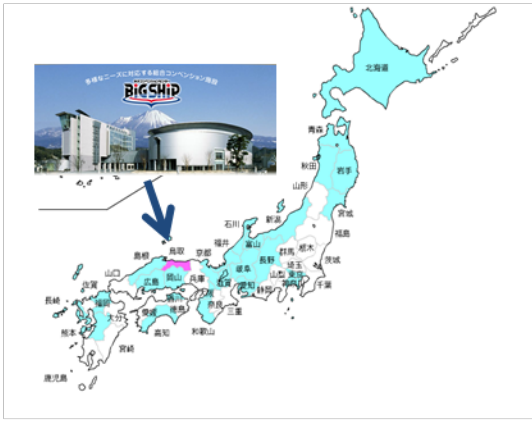
SS2016プログラム委員長
小笠原 秀人 (東芝)・大平 雅雄(和歌山大学)

【開催要領】
◆日程: 2016 年 6 月 5 日 (日)～ 8 日 (水)
6 月 5 日 (日): 併設イベント
6 月 6 日 (月)～ 8 日 (水): 本会議
◇場所 (本会議):米子コンベンションセンター BIG SHIP
◆場所 (併設イベント): 未定

【論文投稿スケジュール】
◆投稿締切: 2016 年 3 月 4 日 (金) ※この場で応相談
◇採否通知: 2016 年 4 月中旬 (予定)
◆最終原稿締切: 2016 年 5 月 20 日 (金)

【募集要領】
◆研究論文 (A4 版 5 ページ以上 10ページ以内)
◇Future Presentation (A4 版 1～ 2ページ)
◆経験論文 (A4 版 5 ページ以上 10ページ以内)
◇事例報告 (要旨 A4 版 1ページと，
スライド原稿 12 枚～ 18 枚)

SS のこれまでの開催地



ソフトウェア・シンポジウム（SS）とは

第 1 回目のソフトウェア・シンポジウムは，1980 年の末に開催された．主催は，ソフトウェア産業振興協会 (当時) の技術委員会 (SIA/TC) であった．通常のアカデミックなコンファレンスとは違って，開発現場で働くソフトウェア技術者たちが，自らの経験を通じて獲得した実践的な技術や知識を交流する場が必要だという認識は，この委員会の発足当時から強く存在していたのだが，ある技術調査のためのワーキング・グループ活動をきっかけとして，その成果発表を兼ねたイベントとして，このシンポジウムが企画されたのである．それからほぼ 1 年後の 1982 年冬には，第 2 回目がやはり同様なコンテクストで開催された．この 2 回のプロトタイピングの成功にもとづいて，翌 1983 年以降のソフトウェア・シンポジウムは，毎年 6 月に定期的に開催するようになった．

ソフトウェア技術者協会 (SEA，「シー」) は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者あるいは研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985 年 12 月に設立しました．

分科会活動

- 環境分科会 (SIGENV)
- 教育分科会 (SIGEDU)
- ソフトウェアプロセス分科会 (SPIN)
- フォーマルメソッド分科会 (SIGFM)
- ソフトウェア品質保証分科会 (SigSQA)

支部活動

- 名古屋支部
- 関西支部
- 九州支部
- 上海支部
- 横浜支部
- 東北支部
- 北陸支部
- 広島支部
- 盛岡支部

ソフトウェア・シンポジウム 2016
論文集

– 2016 年 6 月 12 日　初版発行

編者　　小笠原秀人
　　大平雅雄

発行者　ソフトウェア技術者協会

〒157-0073 東京都世田谷区砧二丁目 17 番 7 号
株式会社ニルソフトウェア内
ソフトウェア技術者協会　事務局
TEL: 03-6805-8931
http://sea.jp/

ソフトウェア技術者協会